Università degli studi di Modena e Reggio Emilia

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

DataRiver: un sistema di integrazione dati Open Source

Relatore:

Prof.ssa Sonia Bergamaschi

Candidato:

Francesco Nigro

Anno accademico 2006/2007

Indice

Parole Chiave:

Data Integration

MOMIS

Open Source

Data Warehouse

ETL

1 Introduzione

La disponibilità di sistemi di archiviazione sempre più capienti in concomitanza con l'esplosione del numero di attività che vengono informatizzate sta ponendo nuove sfide nella gestione delle informazioni. Già oggi una azienda di medie dimensioni può trovarsi costretta da un'infrastruttura informatica poco flessibile che non permette di accedere velocemente e facilmente come si vorrebbe alle informazioni, con il pericolo concreto di avere enormi volumi di dati che vengono sprecati perché non raggiungibili.
Nella maggior parte dei casi vengono sviluppate internamente alle aziende soluzioni ad hoc che consentono di scavalcare i limiti comunicativi tra diverse sorgenti di dati, con il risultato di ottenere infrastrutture intricate, fragili e poco flessibili.
Una soluzione che si sta affermando velocemente per permettere di trarre nuove informazioni dai dati già esistenti in azienda, nonché nello scenario più ampio di internet, è la Data Integration; sotto questo termine vengono raggruppate tutte le tecniche che consentono di aggregare diverse sorgenti dati e di metterle in relazione sviluppando una nuova e più completa visione.
Nell'arco di circa una decina d'anni è stato sviluppato presso l'Università di Modena e Reggio Emilia MOMIS, un progetto particolarmente avanzato per la Data Integration che permette di mappare più sorgenti dati in una vista virtuale globale; creata la vista virtuale è poi possibile interrogarla ottenendo come risposta la fusione dei dati provenienti da tutte le sorgenti mappate.
Scopo della tesi è valutare l'evoluzione di MOMIS investigando in particolare i vantaggi che si otterrebbero dall'eventuale rilascio del progetto sotto una licenza Open Source.
Per prima cosa è stata effettuata un'analisi sintetica del mondo della Data Integration per evidenziare lo stato dell'arte attuale dei sistemi di integrazione nel mondo commerciale e Open Source e per tracciare, a grandi linee, la situazione economica di questo mercato, individuando i vendor di

maggior rilievo.
Per decidere la strategia evolutiva, se adottare una licenza Open Source e cosa effettivamente rilasciare, è stata effettuata un'analisi di MOMIS sia dal punto di vista delle funzionalità offerte, per metterne in luce punti di forza ed eventuali migliorie da apportare, sia dal punto di vista strettamente tecnico e architetturale, per vagliare l'effettiva realizzabilità di una eventuale separazione di parti del sistema da rilasciare alla community.
E' stata inoltre effettuata l'analisi approfondita del mondo Open Source, vagliando le diverse licenze, le strategie commerciali più diffuse, le diverse dinamiche di sviluppo e i vantaggi o svantaggi pratici che questa scelta comporta.
Infine sono state confrontate due soluzioni Open Source per la Data Integration già affermate, Talend e XAware, mettendo in luce le diverse architetture, funzionalità e contesti di utilizzo.
Nella tesi del collega Fabio Romano Analisi e valutazione comparativa dei principali sistemi di integrazione dati commerciali rispetto al sistema Momis attraverso il benchmark Thalia è stata inoltre effettuato il confronto con piattaforme sviluppate da grandi player come: IBM, Oracle e Microsoft.

La tesi è così articolata:
Nel **Capitolo 2** viene presentato brevemente il mondo della Data Integration mettendo in risalto le motivazioni che stanno spingendo i clienti ad adottare soluzioni per l'integrazione, le diverse strategie adottabili per integrare e la situazione commerciale del mercato dell'integrazione.
Nel C**apitolo 3** viene presentato MOMIS ponendo l'accento sull'architettura software del sistema.
Nel **Capitolo 4** vengono affrontate in profondità le tematiche relative all'Open Source riguardo alle basi dell'Open Source, del Free Software e alle licenze software.
Nel **Capitolo 5** vengono affrontati gli aspetti organizzativi dei progetti Open Source che si discostano talvolta fortemente dai modelli classici chiusi.
Nel **Capitolo 6** vengono mostrati i vantaggi concreti del modello Open Source

visto sia dalla parte del cliente che acquista il software che da parte del produttore.
Nel **Capitolo 7** viene introdotto DataRiver, il progetto nato per rendere MOMIS Open Source, in cui vengono considerati tutti i diversi step per arrivare al rilascio pubblico del progetto.
Nel **Capitolo 8** infine vengono analizzati i due prodotti Open Source di maggior successo per la Data Integration.
Infine nel **Capitolo 9** vengono presentate sinteticamente le conclusioni.

2 Data Integration: principi e sistemi

2.1 Cos'è la Data Integration

Che il volume delle informazioni che ci troviamo a trattare sia esploso e che sia destinato inesorabilmente ad aumentare è risaputo e non è pertanto necessario dilungarsi su questo argomento; IDC prevede una crescita costante del volume dei dati che le aziende si troveranno a gestire intorno al 50% annuo, almeno fino al 2010. Basta considerare anche solo la digital footprint che ogni persona genera ogni giorno per rendersi conto della proporzione del fenomeno. L'aspetto interessante, che verrà trattato, non è tanto questo trend quanto le sfide che pone nella gestione e nell'accesso ai dati.
E' evidente come il problema della frammentarietà delle sorgenti dati, e della difficoltà nell'accesso delle stesse, sia attuale e particolarmente sentito nel mondo aziendale dove è facile doversi destreggiare tra diversi sorgenti dati e database realizzati in tempi diversi, per progetti diversi e con tecnologie diverse.
Per salvaguardare il budget e riuscire a trarre un vantaggio dal patrimonio di informazioni che l'azienda possiede già, gli strumenti di data integration sono diventati una soluzione molto diffusa.
Con data integration si è soliti indicare tutte quelle attività che permettono di unificare diverse sorgenti dati in una unica, utilizzabile dall'utente in maniera uniforme; benché la definizione sia semplice, nella realtà esistono una moltitudine di strumenti che agiscono in maniera diversa, con risultati e scopi differenti, ma che vengono tuttavia raggruppati nella grande famiglia della data integration.

Gli approcci alla data integration sono diversi ma quelli più diffusi sono principalmente due: in breve, in un caso raccogliamo tutti i dati e li

riversiamo all'interno di una nuova struttura uniforme e coerente; nell'altro creiamo invece una sovrastruttura logica unificante che permette di utilizzare le sorgenti dati come fossero una sola, suddividendo le interrogazioni degli utenti su tutte le sorgenti ed aggregando poi le singole risposte.
La prima soluzione consiste nell'attingere dalle diverse sorgenti utilizzando dei tool e dei wrapper per poi unificare e aggregare tutti i dati dopo che questi sono stati opportunamente manipolati. Tutti i problemi riguardanti le sorgenti come: tipi dei dati, significato dei dati, dati duplicati e conflitti sono risolti all'inizio della fase di integrazione; i dati così scremati possono quindi essere riversati in una nuova struttura che permetta di accedere alle informazioni in maniera efficiente e rapida.
Quest'idea è materializzata frequentemente in un data warehouse, alimentato tramite strumenti ETL (extract, transform e load, non necessariamente in questo ordine).

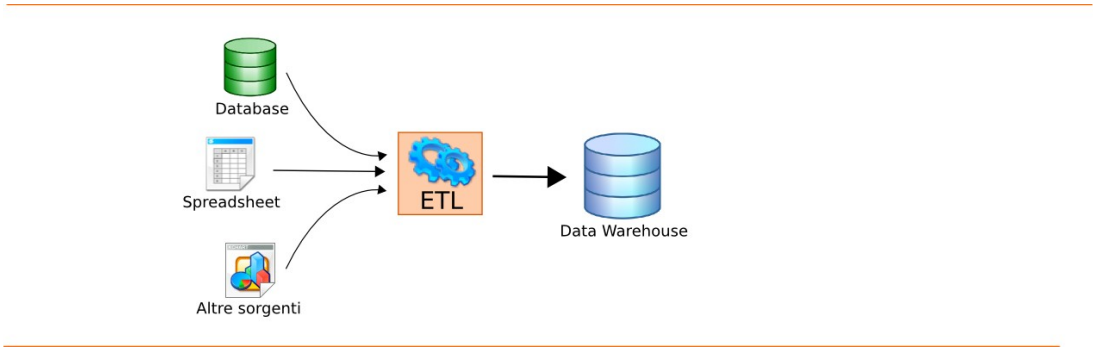


Figura 1: Data Integration attraverso strumenti ETL

Talend, strumento Open Source che verrà analizzato più avanti in questo elaborato, segue questa strategia.
I vantaggi e gli svantaggi sono quelli tipici di un data warehouse: la replicazione dei dati è onerosa, sia in termini di spazio disco occupato che in termini di tempo; normalmente l'operazione di feeding del data warehouse è effettuata offline, col risultato di avere dati non aggiornati per le ricerche.

Inoltre, il data warehouse è un nuovo servizio e come tale va mantenuto e gestito.
Nel caso non ci siano problemi di spazio e i dati, o non vengano aggiornati frequentemente o vengano aggiornati ad intervalli regolari e conosciuti, le limitazioni più penalizzanti di questa scelta sono aggirate; inoltre in uno scenario accademico/scientifico di raccolta di campionamenti, o comunque di dati Write Once Read Occasionally (WORO), l'aggiornamento potrebbe essere eseguito in maniera pianificata e incrementale e quindi richiedere solo un piccolo onere aggiuntivo.
Lo svantaggio di questo approccio è anche il suo punto forte: operare su una replica del database permette di lavorare con la sicurezza di non danneggiare i dati né di causare cali delle performance del sistema, come può facilmente accadere a causa delle query massive richieste per fare data integration.

L'alternativa a questa metodologia, indicata anche col termine federation , consiste nel mantenere tutte le sorgenti dati distinte e nel creare una sovrastruttura logica, risultato della fusione di tutte le sorgenti a disposizione, e nell'attingere alle informazioni attraverso di essa ; il risultato è, per semplificare, un database virtuale perché effettivamente non contiene dati, che l'utente può interrogare in maniera consistente.
Le query che l'utente esegue in realtà vengono suddivise dal sistema in sotto-query inviate poi a ciascuna sorgente dati in maniera indipendente; le singole risposte vengono poi riassemblate in una nuova risposta globale, fornita all'utente.

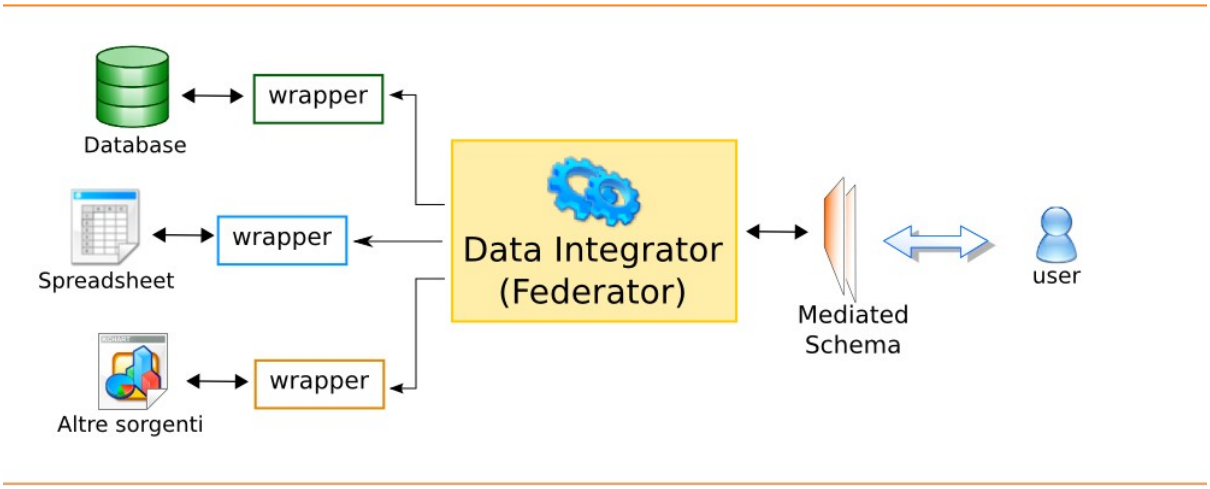
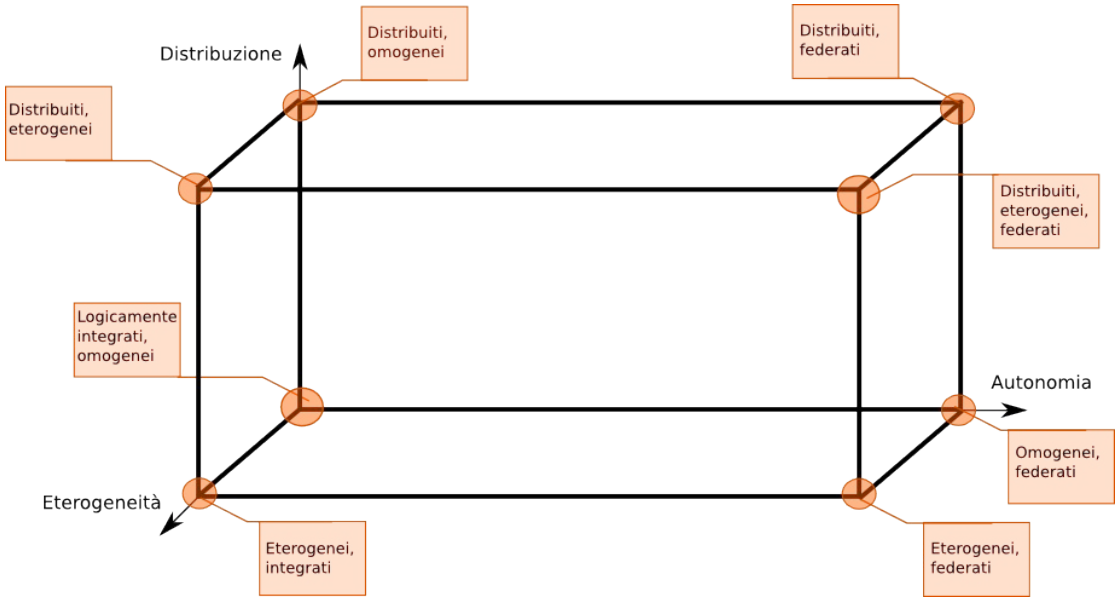


Figura 2: Data Integration tramite Federation

Come si può intuire quest'ultima metodologia risulta più complessa rispetto all'approccio ETL, ma offre anche grossi vantaggi, e per questo si sta diffondendo rapidamente.
Tutti i problemi, visti prima, che insorgono sui dati vengono risolti durante la costruzione dello schema globale, anche se i dati vengono poi effettivamente elaborati e scremati solo nel momento di esecuzione di una query.
Questa soluzione permette di tenere distinte e autonome le varie sorgenti, senza la necessità quindi di andare ad agire sulle strutture e sulle abitudini lavorative consolidate; questo, oltre che corroborare la consueta resistenza al cambiamento delle aziende, permette di mantenere l'infrastruttura esistente pressoché inalterata, con grande gioia per il budget.
In ultima analisi la federation risulta vantaggiosa, se non l'unica strada percorribile, se si ha la necessità di avere dati sempre aggiornati (o non si vogliono affrontare i costi di gestione di un data warehouse) e se soprattutto si vuole mantenere l'autonomia delle sorgenti. E' inoltre molto vantaggiosa se l'integrazione dinamica dei dati coinvolge grandi volumi. In questo caso, infatti, il carico di lavoro può essere distribuito sulle varie sorgenti coinvolte..
La federation è in realtà solo una delle famiglie dei database distribuiti, in cui tutte le sorgenti dati fanno capo ad una unica entità responsabile, che sia un'azienda piuttosto che un'università, che ha la possibilità di agire sulle

sorgenti stesse.
Cataloghiamo i sistemi database distribuiti in base a tre parametri: distribuzione, eterogeneità e autonomia.
Da un lato abbiamo i sistemi logicamente integrati e omogenei, dal lato opposto sistemi distribuiti, eterogenei e federati.



2.2 Perché integrare

Per prima cosa va introdotta la distinzione tra dati e informazione, necessaria per comprendere poi tutto il resto.
Un dato di per se ha valore pressoché nullo, è evidente a chiunque che avere una lunga lista di numeri avulsi dal conteso non ha alcun significato; se questa sequenza la associo però al suo significato, ad esempio, ad una temperatura, e poi ad un luogo e ad una data, ottengo informazione sempre crescente crescente dall'integrazione di diversi dati.
Questo esempio, che può sembrare banale, mette in luce una realtà frequente nelle aziende; capita spesso di incontrare sistemi informativi che gestiscono l'anagrafe dei clienti separatamente dalla fatturazione e dall'assistenza; in questo modo è impossibile sapere ad esempio, se non andando a intervenire

manualmente su tutti i sistemi, qual'è la correlazione tra interventi di assistenza richiesti da un cliente e gli acquisti che questo ha effettuato. E' evidente che integrare i dati già presenti in azienda può generare nuova informazione strategica per l'azienda.
Definiamo innanzitutto quand'è che il valore dell informazione decresce fino ad annullarsi:

- quando non è fornita in tempo utile
- quando non viene comunicata ai soggetti opportuni
- quando non è utilizzabile perché non leggibile o comprensibile.

Procedure manuali di estrazione dei dati, problemi di provision e accounting, formati incompatibili sono problemi comuni che tendono a rendere reali tutti i punti illustrati e a distruggere il patrimonio informativo; la Data Integration cerca di sopperire a tutte queste esigenze con il minimo impatto per i sistemi informativi già esistenti.

2.3 Il mercato della Data Integration

I principali e più conosciuti vendor del mondo IT sono entrati o stanno entrando nel mondo della Data Integration, anche in questo campo quindi i protagonisti sono più o meno i grossi nomi che dominano il mercato informatico, con alcune eccezioni.

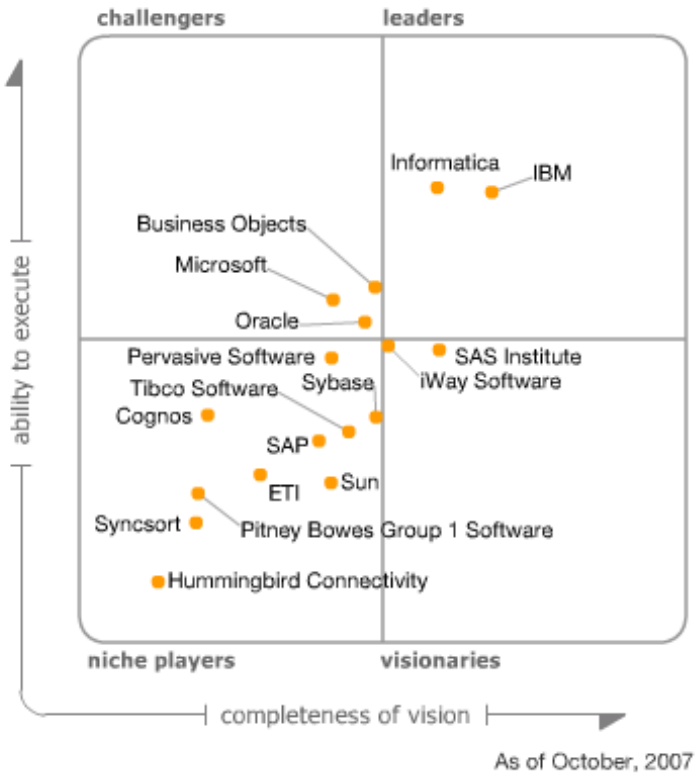


Figura 3: Competitor nella Data Integration

Sopra è riportato il Magic Quadrant emerso dall'analisi del 2007 di Gartner, da cui risulta evidente come i vendor più blasonati siano già entrati, con più o meno successo, in questo mercato; ne è un esempio IBM che domina il mercato per vendite.
Tutti i soggetti riportati hanno in comune la caratteristica di offrire prodotti esclusivamente proprietari e per tanto non verrano trattati (alcuni di questi sono trattati nella tesi di Fabio Romano); verranno valutati infatti solo soluzioni Open Source, per motivi che risulteranno chiari più avanti.
I due principali protagonisti del mercato della Data Integration che offrono prodotti Open Source, Talend e XAware, verrano trattati in dettaglio nei capitoli successivi.

Da notare che seguendo i criteri di Gartner tutti i soggetti che rientrano nel Magic Quadrant generano almeno 20 milioni di dollari l'anno dalla Data Integration e sono presenti in almeno due aree geografiche maggiori,

tipicamente nord America e Europa.
Una trattazione completa di questa analisi va oltre lo scopo del presente elaborato, andando più nel dettaglio ad ogni modo vediamo che i parametri di misura dell'analisi si dividono in Ability to Execute e Completeness of vision. La Ability to Execute rappresenta la capacità concreta di operare in base alla Completeness of Vision che è la visione che l'azienda detiene del mercato e di se stessa nel mercato.
La **Ability to Execute** si basa sulla valutazione di una serie di parametri tra cui: caratteristiche tecniche offerte dai vari prodotti (le feature offerte dai programmi, qualità dei programmi, stato di sviluppo ecc), condizione finanziaria e organizzativa dell'azienda (liquidità, piani di investimenti nel settore e condizione finanziare globale dell'azienda), politiche di pricing, commercializzazione e qualità del supporto pre-sales, efficacia del marketing (chiarezza, efficacia nel influenzare il mercato, valore del brand), supporto al cliente (assistenza e consulenza), capacità di raggiungere gli obbiettivi prefissati.

Tra i parametri più importanti invece per la **Completeness of Vision** si trovano:

capacità di comprendere le necessita del mercato e di rispondere a queste necessità con prodotti e servizi adeguati, strategia di marketing, strategia di vendite, offerta commerciale (linea di prodotti), il modello di business, modello di innovazione, strategia geografica (commercializzazione in altri paesi, distribuzione geografica della forza lavoro, accordi internazionali).

In base al livello di Ability to Execute e di Completeness of Vision possiamo quindi indicare quattro regioni: la regione dei leader è quella dove entrambe sono massime, abbiamo aziende che hanno una buona visione del mercato e riescono a tradurre questa visione concretamente in prodotti e servizi di buona qualità; la regione dei visionari raggruppa le aziende che hanno una buona Completeness of Vision ma per motivi tecnici, organizzativi o economici non riescono a concretizzare questa visione ad un livello

adeguato.

Le aziende che offrono prodotti e servizi di buona qualità ma non hanno una visione del mercato, inseguono per così dire il passo imposto dai leader e si posizionano nel riquadro dei challenger.

Infine rimangono le aziende di nicchia che si concentrano su qualche particolare aspetto dalla data integration ma non affrontano il mercato nella sua interezza.

Gartner individua IBM e Informatica come i vendor di maggior rilievo nel mercato della data integration; per un trattamento approfondito dell'offerta di IBM si faccia riferimento alla tesi di Fabio Romano Analisi e valutazione comparativa dei principali sistemi di integrazione dati commerciali rispetto al sistema Momis attraverso il benchmark Thalia .

Informatica[m], indicata da Forbes come the next Microsoft , è una delle aziende maggiormente affermate in questo campo e sta continuando ad allargare la sua base di clienti e ad acquisire altre aziende per ampliare la propria offerta. Informatica è stata fondata nella Silicon Valley nel 1993 da due imprenditori: Gaurav Dhillon, ingegnere elettrotecnico che ha lavorato precedentemente per Sterling Software e Unisys, aziende specializzate nella consulenza e nella system integration, e Diaz Nesamoney.
Numerose sono state le acquisizioni di Informatica nel tempo, tra cui: Zimba Software nel 2000, azienda specializzata nella business intelligence, Striva nel 2003, specializzata in software di connettività per mainframe, e Similarity Systems nel 2006, specializzata in data quality.

Tra le soluzione realizzate per gli oltre 3000 clienti di Informatica troviamo il noto produttore di soluzione per lo storage Seagate, per cui è stato sviluppato un sistema centralizzato per la gestione del personale, composto da oltre 44000 impiegati sparsi tra le numerosi sedi nel mondo.

Originariamente Informatica forniva strumenti esclusivamente ETL mentre ora, anche in seguito alle acquisizioni, è in grado di offrire strumenti evoluti per la data migration, data consolidation, data synchronisation oltre che ovviamente per il data warehousing. I principali concorrenti di Informatica secondo Gartner risultano essere proprio le soluzioni Open Source che innescheranno una forte concorrenza sui prezzi delle licenze. Tra i prodotti principali dell'offerta di Informatica ricordiamo: Informatica PowerExchange, la famiglia di prodotti dedicata espressamente alla data integration; Informatica Data Explorer che offre una serie di tool per il data profiling ed è orientato alla costruzione di statistiche per il management; Informatica Data Quality che offre soluzioni per la qualità dei dati come ad esempio duplicate detection; Informatica Complex Data Exchange per l'estrazione e la trasformazione di sorgenti dati semi-strutturate come PDF, Excel ecc. Tutti i prodotti di Informatica infine sono disponibili per Windows e alcune versioni di UNIX.

Il cuore dell'offerta di Informatica è però PowerCenter, una piattaforma unificata di livello enterprise per la Data Integration, in due configurazioni: Standard Edition, molto vicina agli strumenti ETL classici, e Advanced Edition che permette, tramite Data Maps che agisce come una view SQL, di avere accesso in real time a differenti database.

Punti cardine per cui Informatica viene molto apprezzata dai suoi clienti, sempre secondo un sondaggio di Gartner, sono inoltre la regolarità dello sviluppo e la qualità del supporto.

Non è stato possibile reperire sufficienti informazioni per effettuare un'analisi puntuale dei prodotti, non essendo Open Source non vengono forniti dettagli tecnici ma solo informazioni commerciali; per poter leggere i white paper o qualsiasi documento di approfondimento tecnico è infatti necessario essere un cliente o abbonarsi alla developers network[n].

Qui di sotto è riportata una tabella riassuntiva delle caratteristiche più

salienti di PowerCenter.

Enterprise Applications Software as a Service (SaaS)	Databases and Data Warehouses	Messaging Systems	Technology Standards	Complex Data (*)
JD Edwards EnterpriseOne	Adabas ✓	JMS ✓	Email (POP, IMAP)	EDI ✓
JD Edwards World	C-ISAM	MSMQ ✓	HTTP(s) ✓	HL7 ✓
Lotus Notes	Datacom ✓	TIBCO ✓	LDAP ✓	SWIFT ✓
Oracle E-Business Suite ✓	DB2 ✓	webMethods Broker ✓	Web services ✓	ACORD ✓
PeopleSoft Enterprise	Essbase	WebSphere MQ ✓	XML	HIPAA ✓
Salesforce (salesforce.com) ✓	IDMS ✓			EDIFACT ✓
	IMS DB ✓			Excel
SAP NetWeaver BI ✓	Informix Dynamic Server			PDF
SAS	JDBC, ODBC			Word
Siebel	Netezza Performance Server			
	Oracle ✓			
	SQL Server ✓			
	Sybase			
	Teradata			
	VSAM ✓			

✓ – Accessibile in real time

* – Fornito da Informatica Complex Data Exchange

Bisogna comunque tener presente che per chiunque voglia entrare solo adesso nel mondo del software commerciale per la Data Integration i soggetti di riferimento sono al momento principalmente 3: IBM, Informatica e Business Objects.

Il mercato è già stato aggredito dai maggiori vendor che stanno ora consolidando le proprie posizioni; chiunque entri ora sul mercato si trova in svantaggio rispetto ai concorrenti, anche se ci sono ancora ampi margini di espansione.
Sempre secondo le stime di Gartner il mercato è cresciuto dal 2005 a un ritmo intorno al 20% annuo, assestandosi a un valore di circa 1,5 miliardi di dollari nel 2007.
Come ci si poteva aspettare il segmento a maggior crescita è quello enterprise, dove è facile trovare grosse realtà, spesso multinazionali, con strutture ad alta complessità e frammentazione.

Il più grande concorrente da battere al momento non è in ogni caso ne IBM o Informatica, quanto lo scripting casalingo realizzato direttamente dai sistemisti in azienda; è stimato che queste soluzione sviluppate ad hoc in house dai sistemisti potrebbero sbloccare un fetta di valore pari alla dimensione attuale dell'intero settore.
I progetti Open Source hanno la caratteristica abbastanza comune di favore la collaborazione e lo sviluppo condiviso e , sulla carta, sembra pertanto la strada più promettente per sbloccare tutto quel potenziale inespresso.
In conclusione, basandosi su queste osservazioni, i desiderata per un nuovo prodotto risultano essere: posizionamento nel segmento enterprise con un buon supporto multilingue, complessità medio-alta del prodotto e come utente target i sistemisti o comunque profili altamente tecnici, supporto a tecniche di scripting diffuse per sfruttare le competenze che spesso si trovano già nel personale in azienda.
Il mercato enterprise è un mercato complesso da aggredire perché è necessario poter garantire supporto e consulenza in maniera abbastanza continuativa nel tempo e geograficamente delocalizzata; questo può essere un grosso scoglio per una start up con risorse e personale limitato.
Questo segmento d'altra parte è un ottimo terreno per lo sviluppo di prodotti Open Source ed un'offerta basata sui servizi; una volta che il prodotto sia stato accettato è quasi scontata la richiesta di servizi accessori come: formazione, consulenza al deployment e alla manutenzione, personalizzazioni di sorta e certificazione del prodotto sui sistemi informativi aziendali.

3 MOMIS

Nel 1997, partendo dai progetti di ricerca INTERDATA e D2I, nasce, ad opera del DBGroup dell'università di Modena, l'idea di realizzare uno strumento in grado di aggregare informazioni provenienti da sorgenti dati eterogenee in modo automatico, così da riuscire a far emergere, dai dati esistenti ma scorrelati, nuova informazione.
Da dieci anni di ricerca accademica e di sviluppo software vero e proprio è emerso il sistema MOMIS, acronimo di Mediator envirOnment for Multiple Information Sources.
MOMIS, partendo da una serie di sorgenti eterogenee, è in grado di costruire una vista globale virtuale dei dati, in inglese Global Virtual View, rappresentazione dello schema unificante di tutte le sorgenti, da cui è possibile estrarre le informazioni risultanti dalla fusione dei dati contenuti nelle diverse sorgenti; con un pizzico di fortuna questa mappatura avviene in maniera automatica e con pochissimo sforzo.
Una volta ottenuta la Global Virtual View (GVV in breve) è possibile utilizzarla (ad esempio interrogarla) come se si trattasse di una nuova sorgente dati omogenea con evidenti vantaggi sia in termini di usabilità che di ricchezza informativa.
Il linguaggio ODLI^3 è mappato in una logica descrittiva OLCD sviluppata dal DBGroup. Questo consente al sistema di poter eseguire attività di reasoning tipiche delle logiche descrittive molto utili nelle scoperte di mapping tra sorgenti dati.
MOMIS si basa su un linguaggio object-oriented, chiamato ODLI^3, che estende l'Object Definition Language (ODL) aggiungendo i concetti di synonym of, broader term, narrower term e related term.

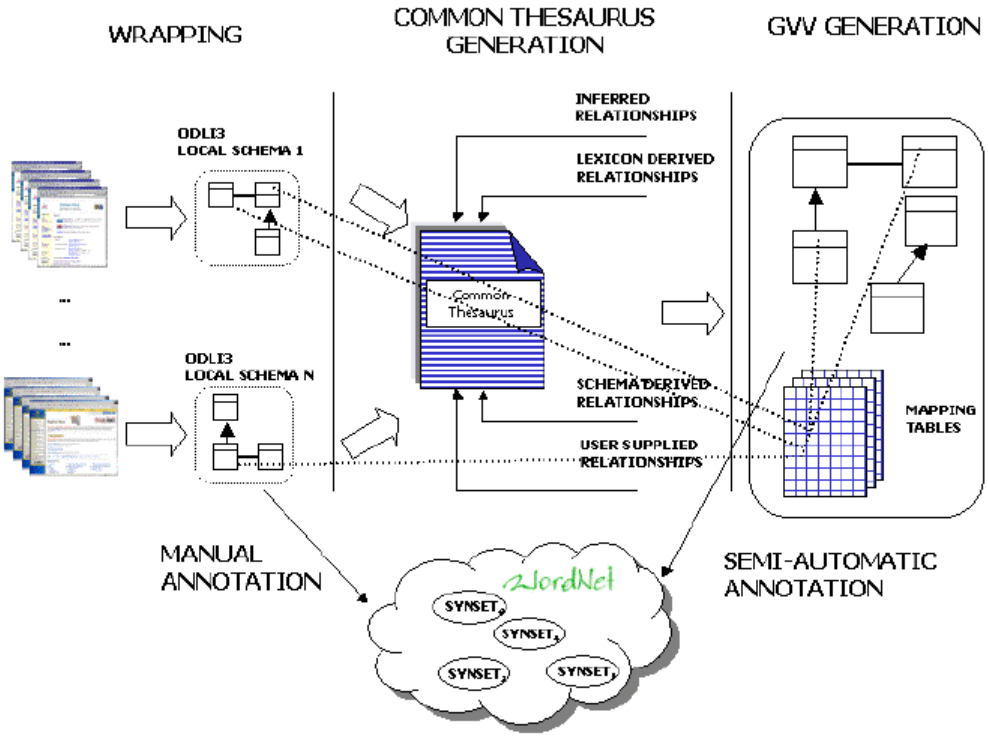


Figura 4: Costruzione della GVV

Vediamo brevemente la successione di passi, mostrati in figura, che portano alla costruzione della GVV.

1. Per prima cosa si estraggono gli schemi della varie sorgenti, facendo uso di appositi wrapper, e li si traduce in ODLI[3]; l'estrazione è immediata nel caso di data base relazionali, mentre per sorgenti XML i file devono prima essere parsati da un wrapper per estrarre lo schema contenuto nei file XML stessi
2. Il progettista quindi associa, con l'aiuto di WordNet, un significato ad ogni elemento dello schema; questa fase serve per chiarire al sistema quali parole, anche se diverse, sottintendono lo stesso significato semantico
3. Utilizzando le annotazioni del passo precedente vengono generate le relazioni intra ed inter schema tra classi e attributi delle sorgenti che vanno a costituire il Common Thesaurus
4. Creazione della GVV
5. Infine, analogamente a quanto già fatto con le sorgenti, si esegue

l'annotazione semi-automatica della GVV, associando un significato semantico a gli elementi globali che compongono la GVV

3.1 Architettura

A grandi linee i componenti fondamentali che costituiscono MOMIS sono il SI-Designer e il Query Manager:
SI-Designer è un programma scritto in Java, così come anche tutti gli altri componenti, che ha il compito di supportare e guidare l' Integration Designer attraverso la creazione della GVV appoggiandosi ad alcuni tool esterni come WordNet, Artemis e ODB-Tools.
Lo schema così creato può essere poi interrogato dall'utente tramite il Query Manager.

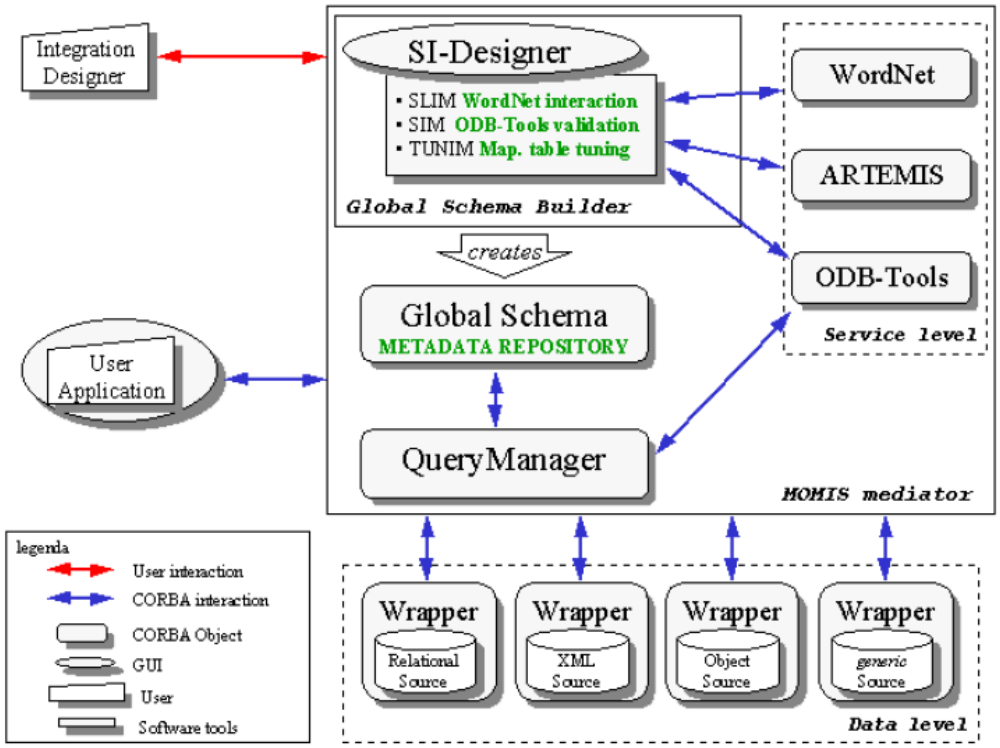


Figura 5: Schema dei componenti di MOMIS

Andando più in dettaglio vediamo che SI-Designer è composto da tre moduli fondamentali:

- SIM
- SLIM
- TUNIM

SLIM, acronimo di Sources Lexical Integrator Module, è il modulo che si occupa di estrarre le relazioni inter-schema appoggiandosi a WordNet;
SIM, acronimo di Source Integrator Module, con l'ausilio di ODB-Tools estrae le relazione intra-schema, effettua il controllo delle relazioni e ne inferisce delle nuove;
TUNIM, acronimo di Tuning of the Mapping table, gestisce la creazione della GVV.
Come si vede in figura MOMIS si interfaccia poi all'esterno a dei wrapper che a loro volta si interfacciano direttamente con i dati.
I wrapper hanno il compito di descrivere gli schemi delle sorgenti, durante la fase di costruzione della GVV, e di tradurre le query che riceve dal Query Manager per le sorgenti dati.
I wrapper attualmente esistente per MOMIS sono per file XML, SqlServer, JDBC generico, JDBC-ODBC e Access.

WordNet è un dizionario lessicale in inglese di circa 150,000 parole sviluppato dalla Princeton University (non verrà pertanto analizzato) che permette di associare ad ogni parola un significato semantico.
WordNet è rilasciato sotto una licenza simil BSD (vedi i paragrafi successivi per una visione dettagliata della licenza BSD) e può essere quindi integrato senza problemi all'interno di altri software, commerciali o Open Source.

ODB-Tools è un framework per la verifica di schemi e per l ottimizzazione di interrogazioni di database object oriented (OODB) sviluppato presso l'Università di Modena.
ARTEMIS infine, acronimo di Analysis and Reconciliation Tool Environment for Multiple Information Source, è un tool sviluppato presso l'Università di

Milano e Brescia che implementa l'analisi semantica e tecniche di clustering delle classi ODLI3 basate su affinità.

La costruzione della GVV è composta da una serie di passi correlati tra loro che vanno eseguiti con un certo ordine; è possibile tornare indietro ma in generale non è possibile passare ad un passo successivo senza aver completato prima quelli precedenti; è evidente che se non configuro prima la connessione alle sorgenti non possono creare il mapping.

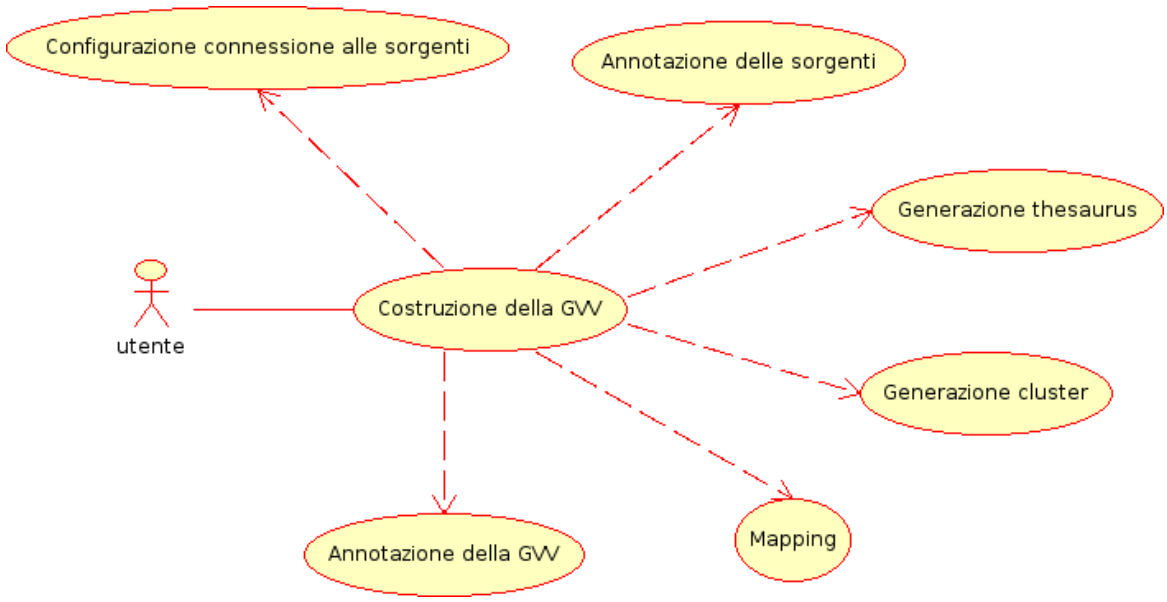


Figura 6: Use case della costruzione della GVV

3.2 Class Diagram

La creazione del class diagram è stata portata avanti con lo scopo di individuare eventuali componenti facilmente staccabili dal resto dell'architettura e che potessero eventualmente tenuti proprietari (vedi capitoli successivi sulla commercializzazione per approfondire).
MOMIS è composto da 1031 classi suddivise in 88 package per un totale di

svariate migliaia di linee di codice, è evidente che riuscire a rappresentare con efficacia il class digram completo è una sfida molto ardua.

Vediamo innanzitutto la gerarchia delle classi più importanti:

3.2.1 Wrapper

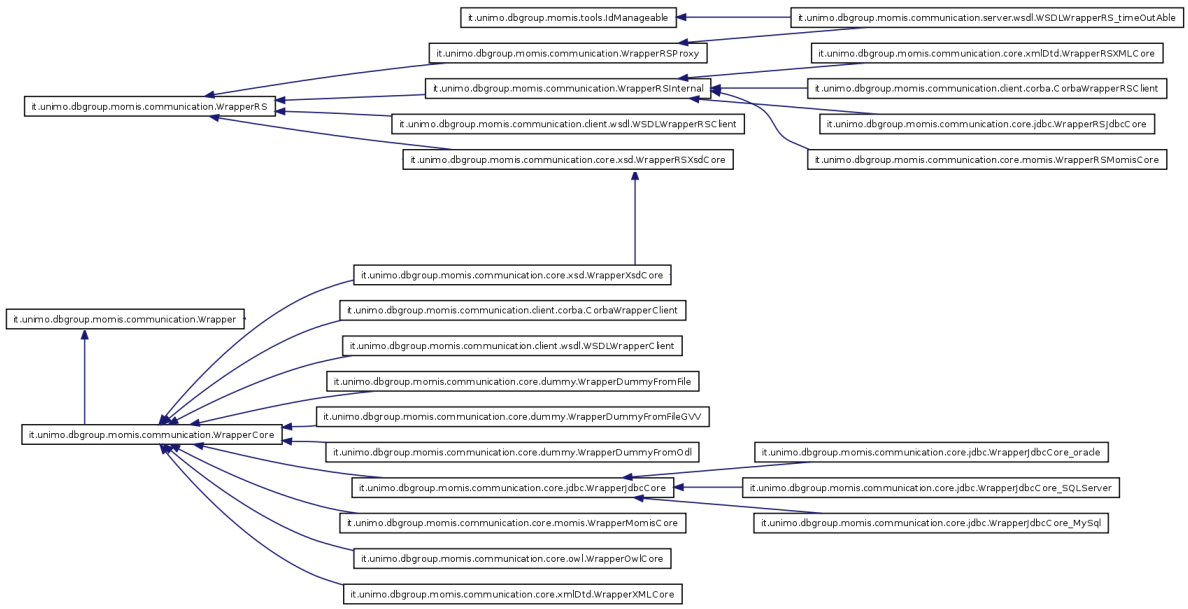


Figura 7: Gerarchia delle classi dei Wrapper

3.2.2 ODLI3 - SIDesigner

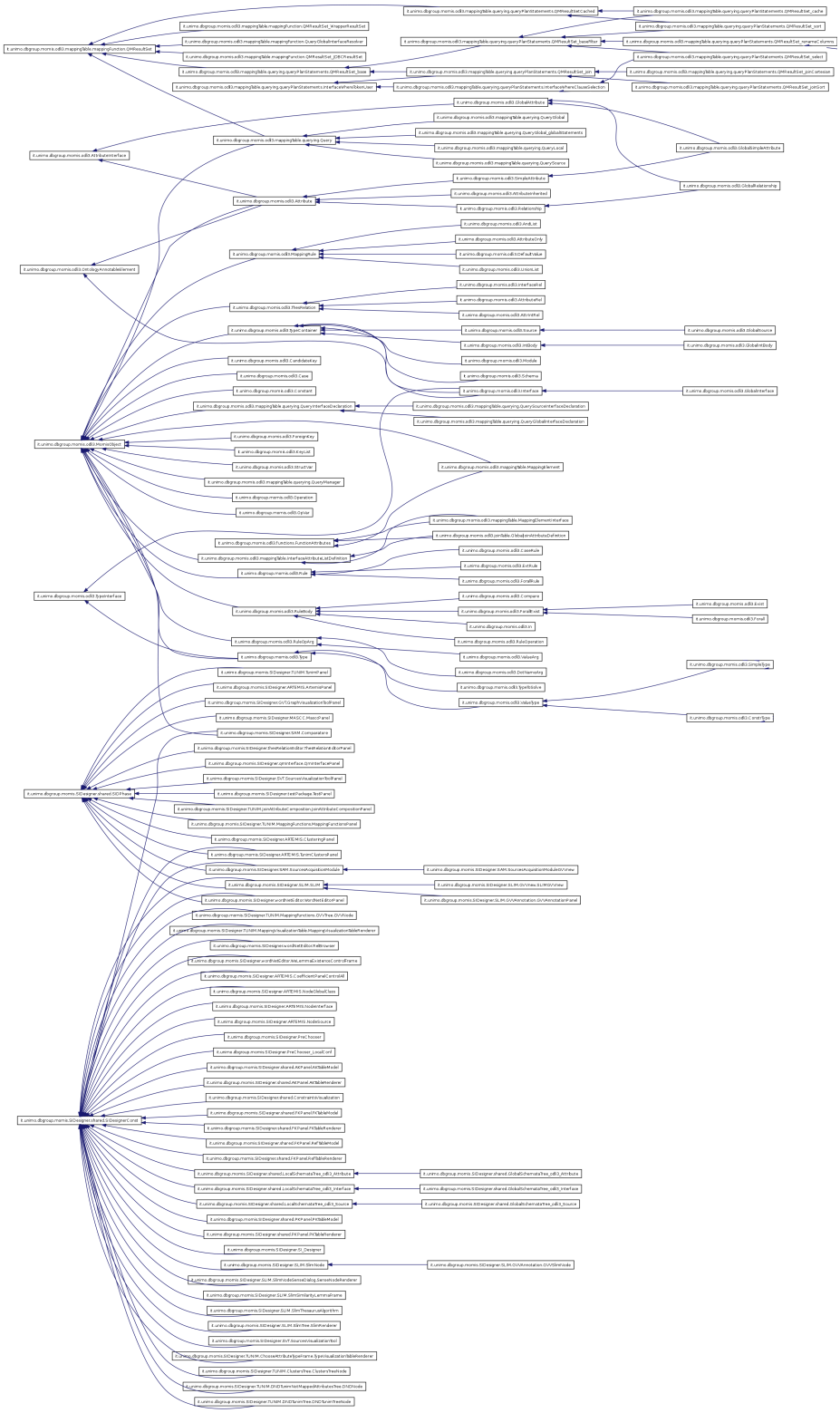


Figura 8: Gerarchia delle classi di ODLI3 e SIDesigner

3.2.3 Oql_Query

La figura sottostante rappresenta la super classe Oql_Query da cui ereditano tutte le Query Oql

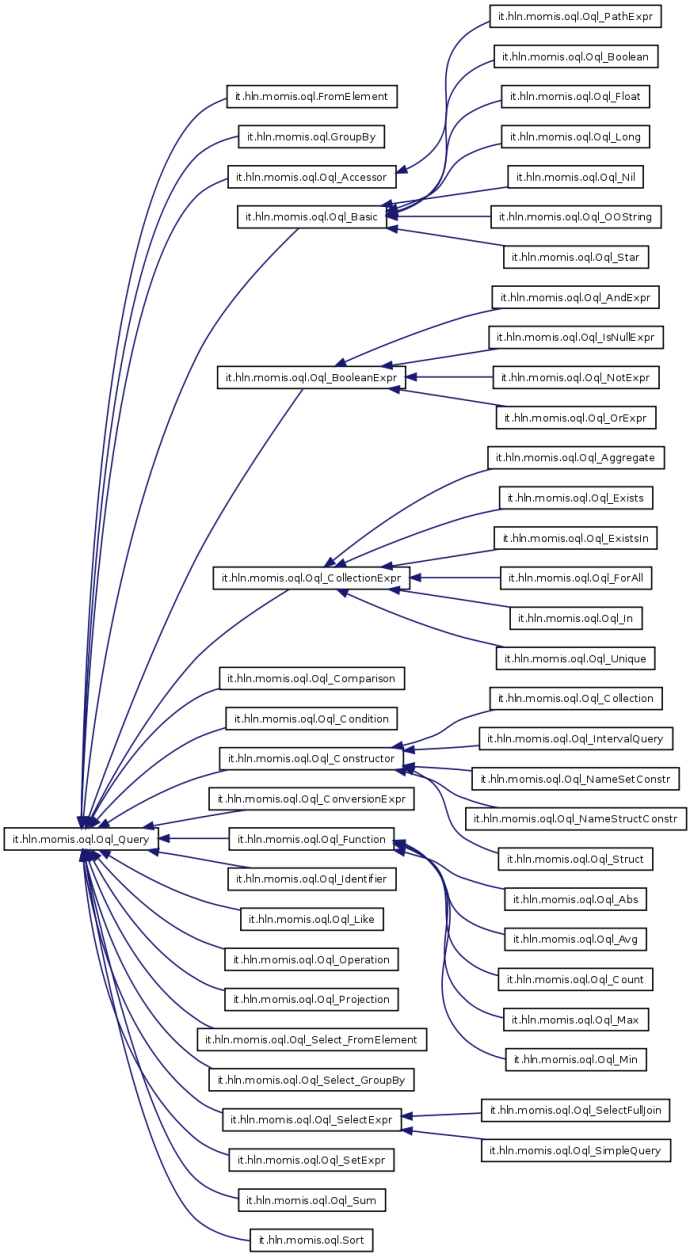


Figura 9: Super classe Oql_Query

Entrando più in dettaglio con il class diagram:

3.2.4 SIM

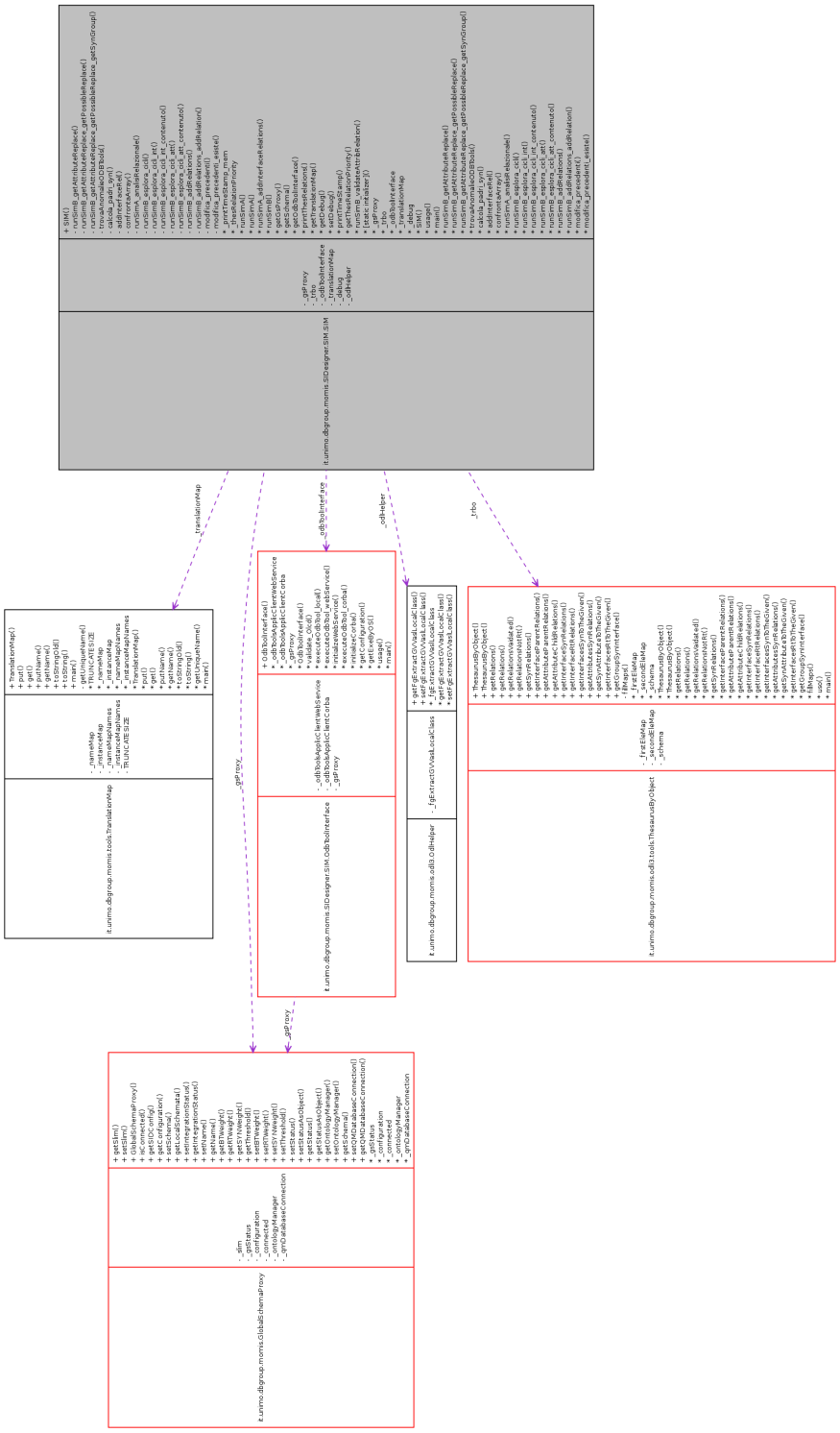


Figura 10: Class Diagram di SIM

3.2.5 SLIM

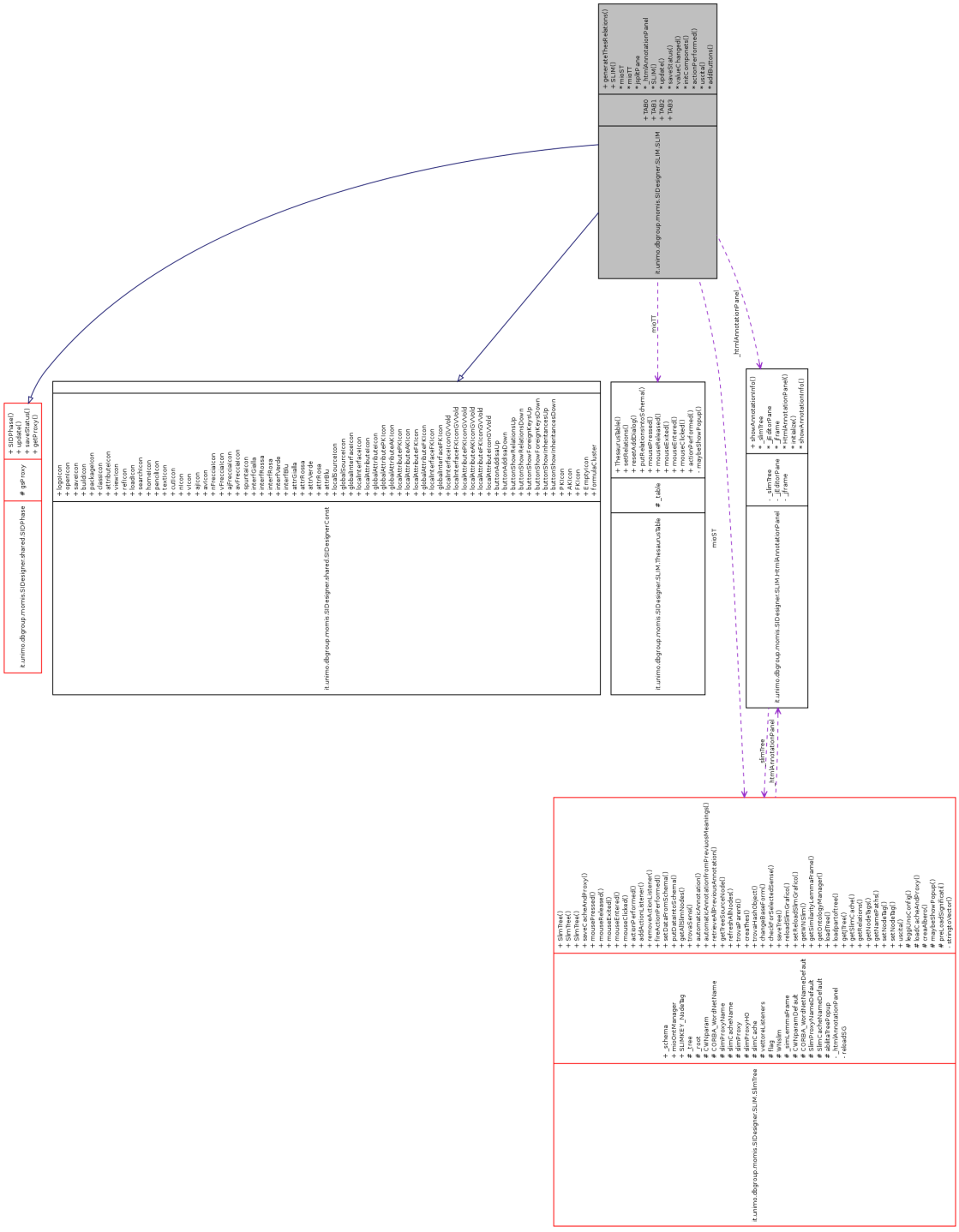


Figura 11: Class Digram di SLIM

3.2.6 TUNIM Panel

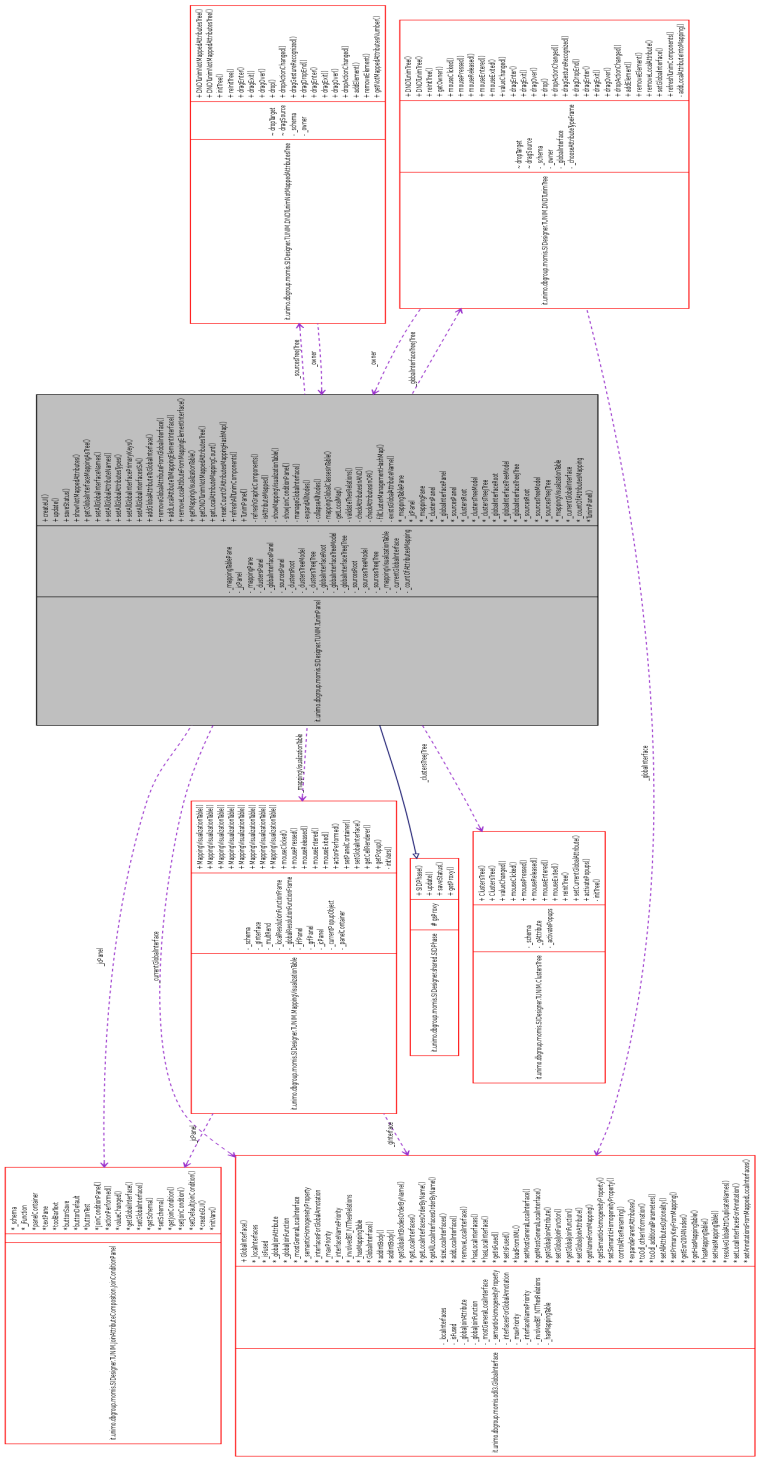


Figura 12: Class Diagram di TUNIM Panel

3.2.7 ARTEMIS

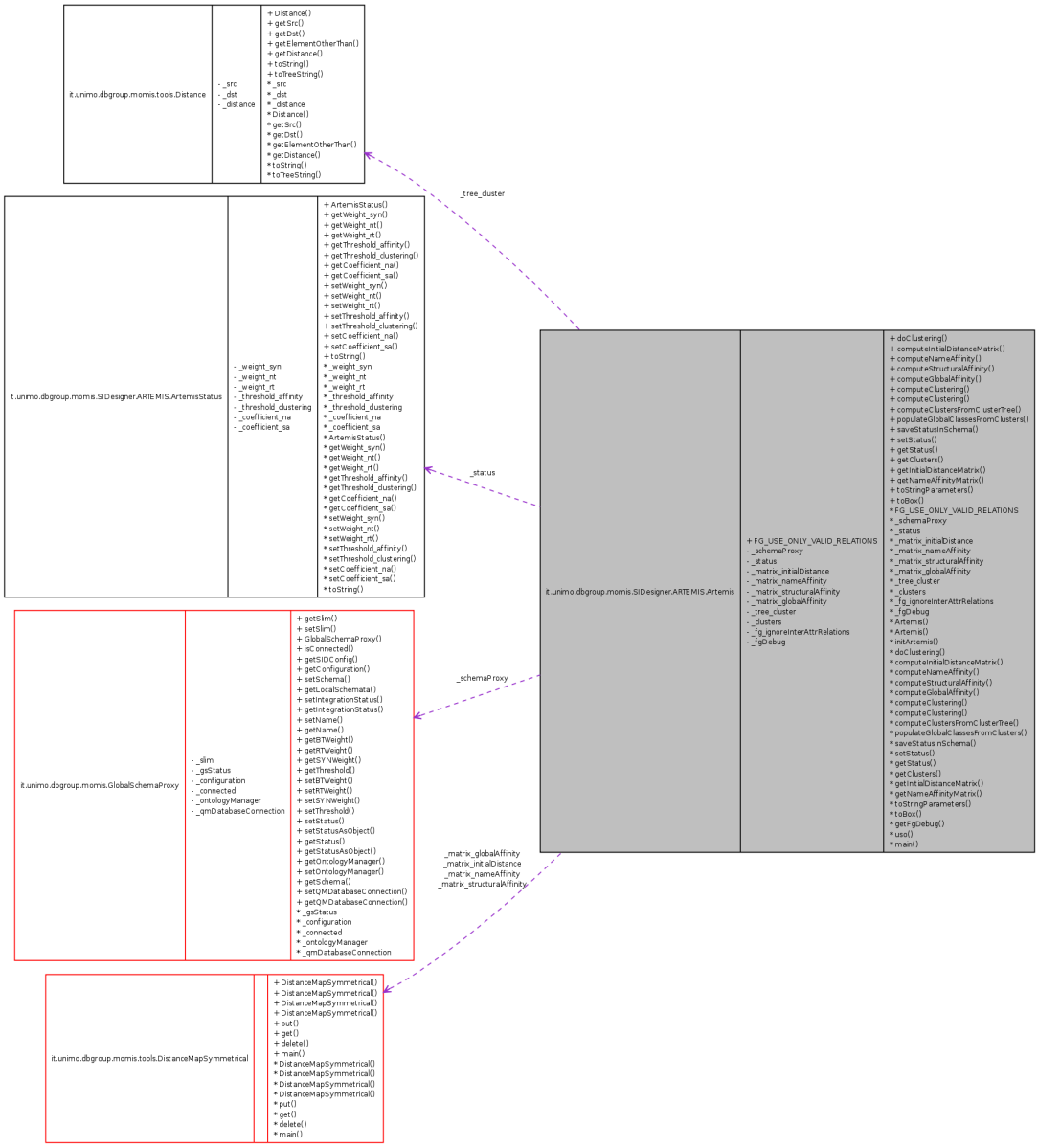


Figura 13: Class Diagram di ARTEMIS

4 L'Open Source e il Free Software

L'Open Source è evoluto dagli anni '90 uscendo dagli scantinati, dove nerd indaffarati sognavano di cambiare il mondo, per diventare un fenomeno internazionale di successo all'interno delle aziende; oggi l'Open Source è una realtà che ha convinto vendor che operano in diversi campi e con interessi completamente differenti ma che hanno trovato in questo modello rivoluzionario di sviluppo ed economico la chiave per il successo sul mercato.

4.1 Free Riding

Il Free Rinding si potrebbe definire come l'utilizzo iniquo del lavoro o delle risorse altrui e viene brutalmente tradotto in italiano con andare a scrocco .

Il sospetto che nasce spontaneo quando si approccia al mondo open è che le ore di duro lavoro spese su un progetto vadano poi effettivamente a rimpinguare le casse di qualcun altro; benché questo sospetto si fondi sul concetto, sfuggevole e personale, di giustizia ed equità, come mostrerò successivamente esiste un limite comunemente accettato abbastanza definito.

A partire dagli anni '80 sono state sviluppate delle licenze che circoscrivono con successo il fenomeno del Free Riding e danno risalto ai vantaggi dello sviluppo condiviso di codice.

Così come per il software proprietario anche per quello open c'è la possibilità di difendersi dal Free Riding tramite, peraltro, gli stessi strumenti condivisi da entrambi i mondi: le licenze.

4.2 Il Free Software

Nel 1983 Richard Stallman (conosciuto con lo pseudonimo RMS), all'epoca giovane ingegnere del MIT, incominciò a lavorare al progetto GNU (leggi gh-nu) con l'obbiettivo di creare un sistema operativo da zero completamente funzionante e soprattutto completamente libero.

L'anno successivo, nel 1984, Stallman, preoccupato che l'università potesse avanzare delle pretese sul progetto GNU che stava sviluppando, abbandonò il laboratorio di intelligenza artificiale del MIT dove lavorava e fondò la Free Software Foundation (in breve FSF) per dedicarsi interamente allo sviluppo del suo sistema operativo libero.

Figura 14: Richard Stallman

Figura 15: Logo del progetto GNU

Quello che all'inizio fece Stallman con il suo progetto GNU non fu altro che scrivere vari programmi per sostituire sistematicamente tutto il software non libero, necessario all utilizzo di un calcolatore, con software libero; il risultato avrebbe dovuto essere un sistema simile a UNIX completo. Giusto come curiosità si può notare che GNU è un acronimo ricorsivo che sta per GNU is Not Unix , GNU non è UNIX , dove l a G di GNU indica l'acronimo stesso.

Il progetto GNU ha prodotto innumerevoli programmi, di cui alcuni fondamentali utilizzati tutt'oggi, come l'editor di testo Emacs e il compilatore GCC.

Stallman creò inoltre una licenza d'uso del software, la GPL, General Public License, sotto i cui termini veniva rilasciato il software libero.

Nel tempo il progetto è progredito e si è sviluppato, portando nuove licenze che verranno illustrate più avanti, ampliando il concetto di Free Software e aumentando, se possibile, la confusione intorno a questo concetto.

Il termine Free Software[1] in inglese è ambiguo perché free ha il doppio significato di libero ma anche gratuito; come si vedrà più avanti il termine è legato esclusivamente alla libertà ed è pertanto da intendere con l'accezione di Software Libero.

La scelta di un nome, simbolo del progetto, che ha bisogno di essere spiegato

dovrebbe far capire immediatamente quanto infelice sia stata tale scelta e che uno studio iniziale di comunicazione sarebbe stato decisamente proficuo.

Ad ogni modo è' importante capire che con Free Software si indica un ben preciso e delimitato insieme di software che garantisce all'utilizzatore 4 precise libertà:

> **Libertà 0:** Libertà di eseguire il programma, per qualsiasi scopo
>
> **Libertà 1:** Libertà di studiare come funziona il programma e adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.
>
> **Libertà 2:** Libertà di ridistribuire copie in modo da aiutare il prossimo.
>
> **Libertà 3:** Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

Semplificando, chiunque può copiare, modificare e ridistribuire, chiedendo un compenso o meno, il software. L'aspetto rivoluzionario per il mercato è la libertà di ridistribuire il software che porta con se innumerevoli risvolti.

Free Software, come si vede, è da legare al concetto di libertà e non alla gratuità, anche se effettivamente la quasi totalità del software libero è anche gratuito; ciò è dovuto al bassissimo costo di distribuzione richiesto dal software oggi giorno, chiunque comprasse una copia di software libero potrebbe condividerlo su Internet rendendolo disponibile in poco tempo a milioni di utenti a un costo irrisorio.

Per ribadire ancora il concetto, Free Software non vuol dire non commerciale, chiunque può vendere Free Software, e un sacco di azienda lo fanno.

Tutto il software che garantisce le 4 libertà elencate sopra è da considerare Software Libero; a livello pratico questo si traduce in una serie di strumenti legali, cioè licenze e termini di copyright.

Un software può essere rilasciato come Software Libero secondo tre diversi schemi generali: dominio pubblico, Software Libero sotto Copyleft, Software Libero non soggetto al Copyleft.

Figura 16: Logo del Copyleft

Tutto il software di **dominio pubblico** è semplicemente privo di copyright, chiunque può utilizzarlo a proprio piacimento senza alcuna restrizione; un software rilasciato sotto dominio pubblico è Software Libero ma le versione future potrebbero non esserlo, chiunque potrebbe utilizzarlo per fare delle versioni proprietarie.

Il **Copyleft**[2] si basa sul concetto di copyright e sulle leggi ad esso legate ma ne ribalta i termini; mentre lo scopo del

copyright è di restringere le libertà di utilizzo di un software, vietandone ad esempio la copia, lo scopo del copyleft è quello di mantenere le libertà date agli utenti.

Il Copyleft impone a chiunque sviluppi software sotto questi termini di mantenere inalterati i termini di rilascio del software; per chiarire, se un programmatore scarica un software rilasciato sotto i termini del Copyleft, lo modifica e vuole ridistribuirlo è obbligato a farlo sotto i termini del Copyleft. Tale caratteristica è detta virale perché tende ad auto-mantenersi e a diffondersi.

Questo vincolo garantisce che anche tutte le versioni future saranno soggette ai termini del Copyleft.

Il lettore attento avrà notato che tutto quello detto è valido solo nel caso si voglia ridistribuire il software; chi volesse modificare il software per esigenze ed utilizzo personale non è tenuto a rilasciare le modifiche ne a renderle pubbliche.

Una nota di colore: il termine è un gioco di parole con Copyright, diritto d'autore, con right che assume il duplice significato di diritto ma anche di destra , e Copyleft, permesso d'autore, con left che assume il significato di lasciato ma anche di sinistra .

L'idea del Copyleft è materializzata in alcune licenze, che non sono altro che testi in legalese che esprimono l'idea descritta sopra; la licenza Copyleft più famosa è senz'altro la GPL.

Il Software Libero oltre che sotto dominio pubblico può essere rilasciato come Software Libero sotto Copyleft oppure come Software Libero non soggetto al Copyleft: la differenza tra i due, ovviamente, è se sono soggetti o meno a questa clausola di protezione delle versioni future.

Il Software Libero non soggetto a Copyleft è sì soggetto a copyright ma non da alcuna garanzia che le versioni future rimangano sotto un regime di Copyleft.

Quando invece il software è rilasciato sotto i termini di una licenza classica il copyright diventa lo strumento per mantenere , il codice sorgente segreto e l'utente non ha il permesso di mettere mano al software ne di ridistribuirlo; in questo caso parliamo di **software proprietario o privativo**.

Come si vede il movimento Free Software nasce con uno spiccato spirito politico e sociale, dove l'obbiettivo perseguito è la libertà degli utenti e non c'è alcun interesse verso le dinamiche economiche o di sviluppo aziendali.

Citando lo stesso Stallman:

> I am working to build a system where people are free to decide their own

actions; in particular, free to help their neighbors [...]

Most people involved with free software say little about freedom, usually because they seek to be more acceptable to business."

4.3 L'Open Source

Il movimento Open Source nasce molto dopo il Free Software, nel 1998, quando viene anche coniato il termine, con la pubblicazione della Open Source Definition, un documento in 10 punti in cui vengono espressi i principi del software Open Source redatto in prima battuta da Bruce Perens. Il termine Open Source venne proposto durante una riunione da Christine Peterson per indicare che il codice sorgente era disponibile, senza però fare riferimento a tutte le implicazioni filosofiche del Free Software. A questa riunione parteciparono di persona o telefonicamente tra gli altri: John Hall, Larry Augustin, Linus Torvalds, Bruce Perens ed Eric S. Raymond, quest'ultimo uno dei maggiori fautori dell'iniziativa Open Source e autore de La cattedrale e il bazar , pietra miliare in cui viene analizzato il funzionamento dello sviluppo del Software Open.

L'Open Source è un movimento che comprende un gran numero di licenze e, per alcuni versi, risulta essere ancora più liberale del movimento Free Software.

La differenzia sostanziale tra Free Software è Open Source è che per l'Open Source il software open e quello proprietario possono convivere, anche se con li tempo la definizione di Free Software si è allargata per diventare quasi indistinguibile dall'Open Source.
Per chiarire: esistono un numero elevato di licenze accettate come licenze Open Source, tra queste le più famose sono:

- Apache Software License
- New BSD license
- Apple Public Source License
- Eclipse Public License
- GPL
- Intel Open Source License
- Mozilla Public License
- Lucent Public License
- PHP License

- Python license
- W3C License

Come si vede tra le licenze Open Source compare anche la GPL, tipica del Free Software.
In definitiva nell'uso comune le due diciture sono usate come sinonimi ma è preferita la dicitura Free Software quando si vuole porre l'accento sull'aspetto sociale e Open Source quando invece si vuole evidenziare l'aspetto economico pratico.

I termini **FLOSS** e **F/OSS**, acronimi di **Free/Libre/Open Source Software** sono stati coniati per evitare di schierarsi in un senso o nell'altro e per indicare il fenomeno nella sua interezza.

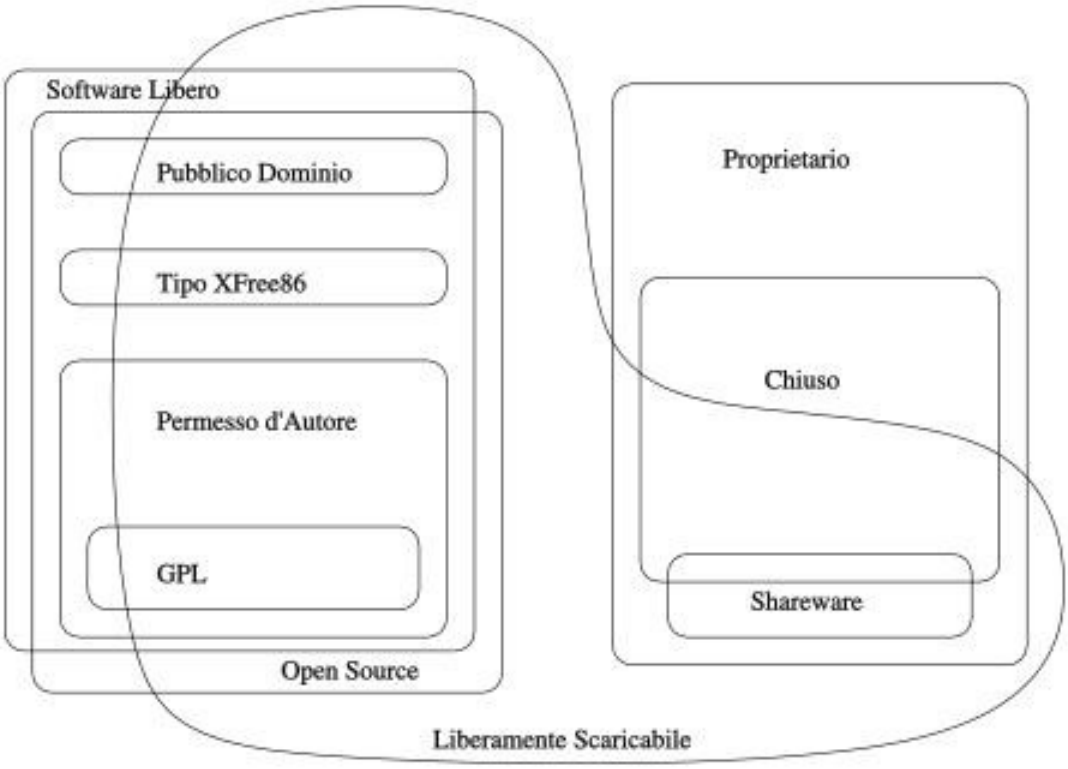


Figura 17: Rapporto tra Free Software e Open Source

Come viene evidenziato nella figura Software Libero e Open Source si sovrappongono largamente.

Alle quattro libertà del Free Software l'Open Source contrappone dieci principi esposti nella **Open Source Definition**[e] riportata qui sotto:

1. Libertà di ridistribuire il software
2. Libertà di visionare il codice sorgente e ridistribuire il software in versione binaria o con il codice sorgente
3. Libertà di modificare il software

4. Integrità del codice sorgente dell'autore
5. Nessuna discriminazione verso persone o gruppi
6. Nessuna discriminazione per i campi di utilizzo del software
7. Distribuzione della licenza
8. La licenza non deve essere specifica di un prodotto
9. La licenza non deve influenzare altro software
10. La licenza deve essere tecnologicamente neutra

1. Libera redistribuzione

La licenza non può limitare alcuno dal vendere o donare il software che ne è oggetto, come componente di una distribuzione aggregata, contenente programmi di varia origine. La licenza non può richiedere diritti o altri pagamenti a fronte di tali vendite.

> Motivo: Imponendo la libera redistribuzione, si elimina la tentazione di rinunciare a importanti guadagni a lungo termine in cambio di un guadagno materiale a breve termine, ottenuto con il controllo delle vendite. Se non vi fosse questa imposizione, i collaboratori esterni sarebbero tentati di abbandonare il progetto, invece che di farlo crescere.

2. Codice sorgente

Il programma deve includere il codice sorgente e ne deve essere permessa la distribuzione sia come codice sorgente che in forma compilata. Laddove alcune forme di un prodotto non siano distribuite con il relativo codice sorgente, deve essere chiaramente indicato il modo per ottenerlo, ad un costo non superiore ad una ragionevole spesa di distribuzione, preferibilmente scaricandolo gratuitamente da Internet. Per codice sorgente si intende la forma in cui un programmatore preferirebbe modificare il programma. Codice sorgente deliberatamente reso illeggibile non risponde ai requisiti. Forme intermedie come l'output di un preprocessore o compilatore non rispondono ai requisiti.

> Motivo: Si richiede l'accesso al codice sorgente poiché non si può far

evolvere un programma senza poterlo modificare. Il nostro obiettivo è rendere facile l'evoluzione del software, pertanto richiediamo che ne sia resa facile la modifica.

3. Prodotti derivati

La licenza deve permettere modifiche e prodotti derivati, e deve permetterne la distribuzione sotto le stesse condizioni della licenza del software originale.

Motivo: La sola possibilità di leggere il codice sorgente non è sufficiente a permettere la revisione indipendente del software da parte di terzi e una rapida selezione evolutiva. Per garantire una rapida evoluzione, deve essere possibile sperimentare modifiche al software e ridistribuirle.

4. Integrità del codice sorgente originale

La licenza può impedire la distribuzione del codice sorgente in forma modificata, a patto che venga consentita la distribuzione dell'originale accompagnato da "patch", ovvero file che permettono di applicare modifiche automatiche al codice sorgente in fase di compilazione. La licenza deve esplicitamente permettere la distribuzione del software prodotto con un codice sorgente modificato. La licenza può richiedere che i prodotti derivati portino un nome o una versione diversa dal software originale.

Motivo: Incoraggiare il miglioramento è bene, ma gli utenti hanno diritto di sapere chi è responsabile del software che stanno usando. Gli autori e i tecnici hanno diritto reciproco di sapere cosa è loro chiesto di supportare e di proteggersi la reputazione.
Perciò, una licenza open source deve garantire che il codice sorgente sia facilmente disponibile, ma può eventualmente richiedere che esso sia redistribuito solo in forma originale più file patch. In questo modo le modifiche "non ufficiali" possono essere rese disponibili pur rimanendo distinte dal codice sorgente originale.

5. Discriminazione contro persone o gruppi

La licenza non deve discriminare alcuna persona o gruppo di persone.

Motivo: Per ottenere il massimo beneficio dal processo, il massima numero di persone e gruppi deve avere eguale possibilità di contribuire allo sviluppo del software. Pertanto viene proibita l'esclusione arbitraria dal processo di persone o gruppi.
Alcuni paesi, inclusi gli Stati Uniti, hanno restrizioni all'esportazione di certi tipi di software. Una licenza conforme all'OSD può avvertire gli utenti di possibili restrizioni e ricordare loro che sono obbligati a rispettare la legge; in ogni caso non può incorporare tali restrizioni essa stessa.

6. Discriminazione per campo d'applicazione

La licenza non deve impedire di far uso del programma in un ambito specifico. Ad esempio non si può impedire l'uso del programma in ambito commerciale o nell'ambito della ricerca genetica.

Motivo: L'intenzione principale di questa clausola è di proibire trappole nelle licenze che impediscano al software open source di essere usato commercialmente. Vogliamo che le aziende si uniscano alla nostra comunità, non che se ne sentano escluse.

7. Distribuzione della licenza

I diritti allegati a un programma devono essere applicabili a tutti coloro a cui il programma è redistribuito, senza che sia necessaria l'emissione di ulteriori licenze.

Motivo: Questa clausola intende proibire la chiusura del software per mezzi indiretti, come un obbligo di sottoscrizione di accordi di non diffusione.

8. Specificità ad un prodotto

I diritti allegati al programma non devono dipendere dall'essere il programma parte di una particolare distribuzione di software. Se il programma è estratto da quella distribuzione e usato o redistribuito secondo i termini della licenza del programma, tutti coloro che ricevano il programma dovranno avere gli stessi diritti che sono garantiti nel caso della distribuzione originale.

Motivo: Questa clausola impedisce un'ulteriore classe di licenze-trappola.

9. Vincoli su altro software

La licenza non deve porre restrizioni su altro software distribuito insieme al software licenziato. Per esempio, la licenza non deve richiedere che tutti gli altri programmi distribuiti sugli stessi supporti siano software open source.

Motivo: I distributori di software open source hanno il diritto di fare le loro scelte riguardo al software che intendono distribuire.

10. Neutralità rispetto alle tecnologie

La licenza non deve contenere clausole che dipendano o si basino su particolari tecnologie o tipi di interfacce.

Motivo: Questa clausola è diretta in particolar modo a quelle licenze che richiedano un gesto esplicito di approvazione da parte dell'utente, al fine di stabilire un contratto. Clausole che richiedano un "click" su interfacce web o di altro tipo possono essere in conflitto con importanti metodi di distribuzione del software, come i siti FTP, le raccolte su CDROM e le copie distribuite sul Web. Tali clausole possono rendere difficoltoso il riutilizzo del software. Le licenze valide devono permettere la possibilità che: 1) il software venga distribuito mediante canali diversi dal Web, sui quali non si possa richiedere un "click" esplicito prima di iniziare il download, e che 2) il programma in oggetto, o sue porzioni, possano essere utilizzare in ambienti privi di interfaccia grafica, nei quali non si possa richiedere la presenza di specifiche finestre di dialogo.

4.4 Freeware, Sharware, Royalty Free

Tutti questi termini stanno ad indicare altre tipologie di software che non hanno niente a che fare ne con l'Open Source ne tanto meno con il Software Libero.
Shareware si riferisce ad un software proprietario che può essere liberamente copiato per poter essere provato; normalmente tali software hanno limitazioni nelle funzionalità o temporali e pertanto, scaduto il periodo di prova, è necessario acquistare una copia completa.
Rilasciare il software come shareware è semplicemente una strategia commerciale, molto diffusa per altro, del tipo prova e poi compra .
Il **Freeware** si riferisce invece al software proprietario che può essere utilizzato gratuitamente; il termine si riferisce esclusivamente al prezzo.
Tutto quello che può essere usato a scopi commerciali, come ad esempio icone, standard tecnici o pezzi di codice, senza la necessità di un pagamento in denaro è definito **royalty free**.

4.5 Le licenze

Lo strumento attraverso cui tutte le idee, i principi e le libertà illustrate operano nel mondo tangibile è la licenza con cui il software viene rilasciato; in questo capitolo verranno introdotte e analizzate brevemente quelle più diffuse.

Si ricorda che è necessario fare riferimento alla versione originale, quasi sempre in inglese, della licenza; le traduzioni qui riportate NON sono ufficiali e potrebbero discostarsi dal significato legale originale.

Come prima cosa è d'obbligo spiegare che il creatore di un software detiene tutti i diritti sull'opera da lui creata e spetta per tanto a lui decidere la

licenza da adottare; questa decisione può essere effettuata e cambiata per ciascuna copia del software ed è pertanto possibile utilizzare licenze diverse per copie diverse.

I termini della licenza si riferiscono ai diritti che l'autore concede a terzi perché l'autore ha ovviamente già tutti i diritti.

Facciamo un esempio pratico: l'azienda PiccoloProgramma sviluppa in casa un editor di testo.

Per il rilascio l'azienda può decidere di fornire lo stesso software sotto più licenze, anche incompatibili tra loro, contemporaneamente.

In effetti è come se l'azienda avesse una grande cornucopia da cui escono le copie del software; per ogni copia l'azienda decide quale licenza apporre ovvero come etichettare quella specifica copia.

Ogni licenza ha effetto solo sulla copia a cui è stata allegata e non ha effetto sulle altre.

L'azienda PiccoloProgramma potrebbe quindi rilasciare l'editor di testo con licenza GPL al pubblico e, contemporaneamente, vendere lo stesso software utilizzando una licenza non open ad un'altra azienda che vuole, ad esempio, integrarlo in un prodotto proprietario.

Un enorme problema insorto con l'affermazione del FLOSS (Free/Libre/Open Source Software, vedi paragrafi precedenti) è il proliferare delle licenze.

Visto che le definizioni per il FLOSS sono di principio, quasi ogni azienda, progetto o gruppo di ricerca che abbia rilasciato un open software di rilievo si è sentita in passato in diritto di creare una nuova licenza specifica per il proprio prodotto.

Questo ha portato celermente ad uno stato di confusione diffuso, con una moltitudine di licenze spesso identiche nella sostanza, obbligando ogni volta alla lettura e all'analisi della licenza per ogni software prima di poterlo utilizzare o modificare.

Solo per citarne alcune, un breve elenco delle licenze considerate compatibili FLOSS:

- IBM Public license
- Microsoft Public License
- Microsoft Reciprocal License
- Nokia Open License
- Intel Open Source License
- Lucent Public License
- Netscape Public License
- Sun Public License
- Apple Public Source License
- NASA Open Source Agreement
- Apache License
- MIT License
- Eclipse Public License
- Mozilla Public License
- BSD License
- QT Public License
- LaTeX Project Public License

Figura 18: Logo dell' Open Source Initiative

Figura 19: Logo della Free Software Foundation

La Free Software Foundation (FSF) e la Open Source Initiative, che sono i due soggetti che certificano quali licenze sono considerate compatibili con il Free Software o l'Open Source, portano avanti da tempo un'opera di sensibilizzazione per fermare il fenomeno.

Benché sia possibile trovare tuttora una licenza personalizzata compatibile FLOSS quasi per ogni azienda di rilievo, ha fortunatamente preso piede l'abitudine di rifarsi alle principali e consolidate licenze FLOSS restringendo la cerchia a meno di una decina, tra cui: GPL, LGPL, GNU Affero, BSD, Mozilla Public License e Apache Software License.

4.5.1 GPL versione 2

La versione 2 della GPL, indicata solitamente come GPLv2, è senz'altro la più diffusa e conosciuta licenza FLOSS e merita un'attenzione speciale, anche se effettivamente dovrebbe essere destinata a lasciare il posto alla più recente versione 3, conosciuta come GPLv3.

La licenza originale e tutte le versioni successive della GPL, acronimo di **General Public License**, sono state create dalla Free Software Foundation, inizialmente per il progetto GNU di Richard Stallman, per poi diventare universalmente accettate.

La GPL è l'emblema del Copyleft e contiene pertanto l'effetto virale che protegge il software anche nelle versioni future; una volta che un software è stato rilasciato con questa licenza qualsiasi modifica apportata dovrà essere ridistribuita sotto tali termini.

Facciamo un esempio: se un programmatore scarica un software rilasciato sotto i termini della GPLv2 può modificarlo o meno, creare una propria versione e rivenderla, purché rilasci questa nuova versione sotto licenza GPL.

Chi riceve il software può modificare il codice ma non la licenza, questo può farlo solo l'autore originale e SOLO sulla parte di codice da lui creata.

Questa licenza risulta l'ideale per stimolare lo sviluppo comunitario e condiviso del software perché mette pienamente al riparo dal free riding (vedi paragrafi precedenti).

Come già detto, l'autore di un software può decidere quale licenza applicare al proprio software e tale decisione ha validità su una singola copia del software; tuttavia una volta che una copia è stata rilasciata sotto licenza GPL tale copia non può più essere ritirata e continuerà a vivere come software open.
Facciamo un esempio: l'azienda PiccoloProgramma sviluppa il suo editor di testo, Parola95, e decide di pubblicarlo con la licenza GPL; i ritorni non sono quelli attesi pertanto viene deciso che la versione successiva verrà rilasciata come software proprietario.
Parola96, che condivide la maggior parte del codice con la versione dell'anno precedente, può essere tranquillamente commercializzato con una licenza non open; in quanto l'autore ne dispone come meglio crede.
La versione precedente, tuttavia, è stata ormai rilasciata sotto GPL e non si può tornare indietro; tale versione continuerà ad esistere e chiunque potrà svilupparla autonomamente senza che l'autore possa proibirlo; le modifiche apportate da terzi sulla versione sotto GPL rimarranno esclusivamente sotto GPL. Le modifica apportate da terzi alla versione GPL rimangono sotto GPL e non potranno quindi essere integrate nella versione proprietaria Parola96; in

questo modo chi partecipa allo sviluppo di un software coperto da GPL è sicuro che il suo lavoro andrà a vantaggio di tutti, e non solo all'autore originario del software.

4.5.2 GPL versione 3

La versione 3 della GPL, indicata come GPLv3, è stata rilasciata nel Giugno del 2007, suscitando non poche polemiche a causa di nuove clausole considerate troppo restrittive per il mondo commerciale.
Benché la Free Software Foundation ci tenga a sottolineare che questa versione è solo un aggiornamento della precedente, le novità introdotte hanno risvolti talmente potenti che considerarla solamente un aggiornamento è decisamente riduttivo.

Figura 20: Logo della GPL versione 3

La versione 3 è incompatibile con le precedenti, **codice rilasciato con GPLv3 non può essere mescolato con codice rilasciato sotto GPLv2**.
Quest'ultimo aspetto è da valutare a fondo prima di decidere quale versione adottare, al momento la versione 2 è senz'altro la più diffusa e lo rimarrà ancora per almeno un paio di anni, adottare nel proprio software la versione 3 potrebbe dire non poter utilizzare una grande quantità di codice già disponibile, risparmiando tempo e denaro, solo per una cattiva scelta sulla licenza.
Un confronto puntuale delle due versione è disponibile nella Appendice A.

Uno sforzo importante è stato impiegato per migliorare l'internazionalizzazione della licenza e l'adattabilità della stessa alle diverse realtà giuridiche; tra le modifiche che potremmo considerare tecniche c'è anche l'introduzione della definizione formale di codice sorgente, così come riportato dalla sezione 1 della licenza:
Il "codice sorgente" di un'opera indica la forma più indicata dell'opera per effettuare modifiche su di essa. Il "codice oggetto" indica qualunque forma dell'opera che non sia codice sorgente.

Al di là di queste modifiche tecniche sono state introdotte molte novità tra cui le più importanti sono:

- Anti Tivoization
- Protezione dai brevetti software
- Regolamentazione delle violazioni e dei tempi di rivalsa

Tivoization è un termine di evidente origine anglosassone che nasce da un caso famoso scoppiato qualche anno fa: la TiVo Inc. ha realizzato un digital video recorder (un apparecchio che è in grado di registrare dalla televisione e salvare i contenuti su hard disk) chiamato appunto TiVo; il software utilizzato per il TiVO contiene codice rilasciato sotto GPL, tra cui anche Linux, e l'azienda ha quindi provveduto a distribuire le modifiche effettuate al codice sorgente, così come stabilito dalla licenza.
Tuttavia anche avendo accesso al codice sorgente non è possibile apporre alcuna modifica a livello pratico perché il TiVo controlla in hardware, tramite chiavi di cifratura, se sono state apportate modifiche al software, rifiutandosi di eseguire qualsiasi cosa che non sia firmato digitalmente dalla casa madre.
Questo stratagemma rende di fatto inutile l'accesso al codice sorgente e ha scatenato molte polemiche perché varca quel confine invisibile considerato come Free Riding.
E' da notare che la casa madre mantiene in ogni caso il diritto di annullare la garanzia del prodotto se questo viene modificato, come normalmente accade anche per tutti gli altri prodotti in commercio.
La sezione 6 è studiata appositamente per mettere al riparto dalla tivoization:
Un "Prodotto Utente" è un (1) "prodotto consumer", cioè qualunque proprietà personale tangibile che è normalmente utilizzata per scopi personali, familiari o domestici, oppure (2) qualunque cosa progettata o venduta per essere utilizzata in ambiente domestico. Nella classificazione di un prodotto come "prodotto consumer", i casi dubbi andranno risolti in favore dell'ambito

di applicazione. Per un dato prodotto ricevuto da un dato utente, "normalmente utilizzato" si riferisce ad un uso tipico o comune di quella classe di prodotti, indipendentemente dallo stato dell'utente specifico o dal modo in cui l'utente specifico utilizza, o si aspetta o ci si aspetta che utilizzi, il prodotto. Un prodotto è un "prodotto consumer" indipendentemente dal fatto che abbia usi commerciali, industriali o diversi da quelli "consumer", a meno che questi usi non rappresentino il solo modo utile di utilizzare il prodotto in questione.

Le "Informazioni di Installazione" per un Prodotto Utente sono i metodi, le procedure, le chiavi di autorizzazioni o altre informazioni necessarie per installare ed eseguire versioni modificate di un programma coperto da questa Licenza all'interno di un Prodotto Utente, a partire da versioni modificate dei suoi Sorgenti Corrispondenti. Tali informazioni devono essere sufficienti ad assicurare che il funzionamento del codice oggetto modificato non sia in nessun caso proibito o ostacolato per il solo fatto che sono state apportate delle modifiche.

Se distribuisci un codice oggetto secondo le condizioni di questa sezione in, o assieme, o specificatamente per l'uso in o con un Prodotto Utente, e la distribuzione avviene come parte di una transazione nella quale il diritto di possesso e di uso del Prodotto Utente viene trasferito al destinatario per sempre o per un periodo prefissato (indipendentemente da come la transazione sia caratterizzata), il Sorgente Corrispondente distribuito secondo le condizioni di questa sezione deve essere accompagnato dalle Informazioni di Installazione. Questa condizione non è richiesta se né tu né una terza parte ha la possibilità di installare versioni modificate del codice oggetto sul Prodotto Utente (ad esempio, se il programma è installato su una ROM)

La condizione che richiede di fornire delle Informazioni di Installazione non implica che venga fornito supporto, garanzia o aggiornamenti per un programma che è stato modificato o installato dal destinatario, o per il Prodotto Utente in cui esso è stato modificato o installato. L'accesso ad una rete può essere negato se le modifiche apportate impattano materialmente

sull'operatività della rete o se violano le regole e i protocolli di comunicazione attraverso la rete.
Il Sorgente Corrispondente distribuito, e le Informazioni di Installazione fornite, in accordo con questa sezione, devono essere in un formato che sia pubblicamente documentato (e con una implementazione pubblicamente disponibile in formato di codice sorgente), e non devono richiedere speciali password o chiavi per essere spacchettate, lette o copiate.

La **protezione dai brevetti software**, problema che per fortuna non è ancora necessario affrontare in Europa, viene assicurata dalla sezione 10; in questa sezione viene infatti chiarito che chiunque distribuisca software con licenza GPLv3 rinuncia a qualsiasi diritto e azione legale per richiedere royalty o risarcimenti per brevetti software:
Licenza Automatica per i successivi destinatari
Ogni qual volta distribuisci un programma coperto da questa Licenza, il destinatario riceve automaticamente una licenza, dal detentore originario del copyright, di eseguire, modificare e propagare il programma, nel rispetto di questa Licenza. Non sei ritenuto responsabile del rispetto di questa Licenza da parte di terze parti.
Una "transazione d' entità" è una transazione che trasferisce il controllo di una organizzazione, o sostanzialmente di tutti i suoi beni, che suddivide una organizzazione o che fonde più organizzazioni. Se la propagazione di un programma coperto da questa Licenza è conseguente ad una transazione di entità, ciascuna parte che ha ruolo nella transazione e che riceve una copia del programma riceve allo stesso tempo qualsiasi licenza sul programma che i predecessori della parte possedevano o potevano rilasciare nel rispetto del paragrafo precedente, e in più il diritto di possesso del Sorgente Corrispondente del programma dal predecessore in interesse, se il predecessore lo possiede o se può ottenerlo senza troppe difficoltà.
Non puoi imporre nessuna ulteriore restrizione sull'esercizio dei diritti garantiti o affermati da questa Licenza. Per esempio, non puoi imporre un prezzo di licenza, una royalty, o altri costi per l'esercizio dei diritti garantiti

da questa Licenza, a non puoi dar corso ad una controversia (ivi incluse le controversie incrociate o la difesa in cause legali) affermando che siano stati violati dei brevetti a causa della produzione, dell'uso, della vendita, della messa in vendita o dell'importazione del Programma o di sue parti.

Per quello che riguarda la **regolamentazione delle violazioni e dei tempi di rivalsa**, l'obbiettivo è stato quello di incentivare un comportamento onesto e favorire una soluzione pacifica delle controversie; qualsiasi soggetto, che violi i termini della licenza, può riottenere interamente i propri diritti, come stabilito dalla licenza stessa, non appena questo interrompa la violazione, a meno che un autore non lo contatti entro sessanta giorni dal termine della violazione.
In questo caso, se è la prima volta che si infrangono i termini della licenza, chi ha violato la licenza ha ancora trenta giorni di tempo per mettersi in regola.

La sezione 8 contiene tutte le indicazioni relative:
Cessazione di Licenza
Non puoi propagare o modificare un programma coperto da questa Licenza in maniera diversa da quanto espressamente consentito da questa Licenza. Qualunque tentativo di propagare o modificare altrimenti il Programma è nullo, e provoca l'immediata cessazione dei diritti garantiti da questa Licenza (compresi tutte le eventuali licenze di brevetto garantite ai sensi del terzo paragrafo della sezione 11).
In ogni caso, se cessano tutte le violazioni di questa Licenza, allora la tua licenza da parte di un dato detentore del copyright viene ripristinata (a) in via cautelativa, a meno che e fino a quando il detentore del copyright non cessa esplicitamente e definitivamente la tua licenza, e (b) in via permanente se il detentore del copyright non ti notifica in alcun modo la violazione entro 60 giorni dalla cessazione della licenza.
Inoltre, la tua licenza da parte di un dato detentore del copyright viene ripristinata in maniera permanente se il detentore del copyright ti notifica la

violazione in maniera adeguata, se questa è la prima volta che ricevi una notifica di violazione di questa Licenza (per qualunque Programma) dallo stesso detentore di copyright, e se rimedi alla violazione entro 30 giorni dalla data di ricezione della notifica di violazione.
La cessazione dei tuoi diritti come specificato in questa sezione non provoca la cessazione delle licenze di terze parti che abbiano ricevuto copie o diritti da te secondo questa Licenza. Se i tuoi diritti cessano e non sono ristabiliti in via permanente, non hai diritto di ricevere nuove licenze per lo stesso materiale, secondo quanto stabilito nella sezione 10.

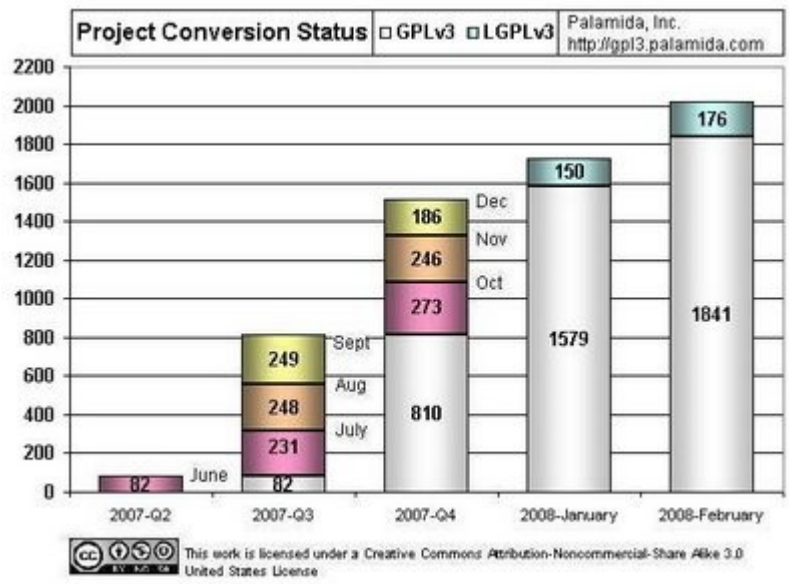


Figura 21: Progetti che adottano la GPL versione 3

Dal grafico riportati qui sopra si nota come il tasso di crescita sia abbastanza elevato ma c'è da aggiungere poi che molti progetti di rilievo, con alcune eccezioni come SAMBA, al momento hanno deciso di stare alla finestra e aspettare che la situazione evolva.
E' probabile che un effetto valanga venga innescato dalla migrazione di alcuni progetti chiave, come il kernel Linux, ma al momento non si può ancora predire quando questo avverrà.

Su circa 6500 progetti coperti da GPL circa 2000 adottano la nuova versione e, ad essere sinceri, la maggior parte di questi non sono di particolare rilievo.

In conclusione la scelta più conservativa al momento è di attendere che si inneschi una migrazione più massiccia.

4.5.3 LGPL

La LGPL può essere vista come una particolare forma di GPL studiata appositamente per le librerie; l'acronimo sta per **Lesser General Public License**, anche se è facile trovare anche la traduzione Library General Public License.

La differenza sostanziale con la GPLv3 è che qualsiasi programma, indipendentemente dalla sua licenza, può linkare le librerie rilasciate sotto LGPL; in questa maniera è possibile per un software non GPL, o anche proprietario, utilizzare tali librerie. La questione è puramente pratica: utilizzare la LGPL significa dare la possibilità ad altri di utilizzare il software senza dare niente in cambio; se questo è un aspetto negativo è da valutare caso per caso.
Le librerie GNU per il C, ad esempio, utilizzano la licenza LGPL perché si ha l'interesse a diffonderne l'utilizzo, sia da software open che proprietari; esistono già librerie alternative per il C non open e utilizzare la semplice GPL proibendone l'utilizzo avrebbe l'unico effetto di far scegliere ai programmi proprietari librerie diverse.
La FSF consiglia, come nell'esempio appena illustrato, di utilizzare la LGPL il

meno possibile, e solo nel caso esistano appunto alternative accessibili ad una libreria open.
Anche la LGPL è stata recentemente aggiornata alla versione 3 portandosi dietro tutte le considerazioni fatte nel paragrafo precedente per la GPLv3.
La LGPLv3 è compatibile con la GPLv3, codice rilasciato sotto queste due licenze può essere utilizzato all'interno dello stesso programma.

4.5.4 Affero

La GNU Affero GPL, o in breve AGPL, introduce alcune clausole che riguardano l'utilizzo in rete del software.

Figura 22: Logo della GNU Affero GPL

Per creare giusto un po' di confusione bisogna sapere che esiste una Affero General Public License, realizzata dalla Affero Inc, che non va confusa con la GNU Affero GPL; le due licenze sono oltretutto simili nei contenuti ma rimangono comunque due licenze distinte.
Vediamo un esempio concreto: come visto, se un programmatore apporta delle modifiche ad un software, è obbligato a renderle pubbliche nel momento in cui decida di ridistribuire il software; come già detto, se non c'è ridistribuzione non c'è l'obbligo di rendere disponibile le modifiche.
Nel caso in cui il software in questione sia un'applicazione web il problema si fa più sottile: renderla disponibile su internet tecnicamente non è ridistribuzione, ma ci è vicino.
E' da considerare una violazione della GPL? A tutti gli effetti no, non c'è violazione della GPL, però si tocca nuovamente su quel confine un po' sfumato del Free Riding.
Per dipanare questo nodo è stata creata la Affero GPL che stabilisce, inequivocabilmente, che rendere accessibile attraverso la rete un software AGPL obbliga a rendere disponibile anche il codice sorgente.
La definizione di distribuzione rimane inalterata, così come stabilito nella

sezione 0:
"Distribuire" un'opera indica qualunque forma di propagazione che permetta a terze parti di effettuare o ricevere delle copie. La mera interazione con un utente attraverso una rete di computer, senza che ci sia alcun trasferimento di una copia, non è considerata "Distribuzione".

Viene altresì introdotta una sezione, la 13, specifica sulla questione:
Indipendentemente da qualunque altra condizione espressa da questa Licenza, se apporti delle modifiche al Programma, la tua versione modificata deve offrire in maniera evidente a tutti gli utenti che ci interagiscono attraverso una rete (se la tua versione supporta tale interazione) l'opportunità di ottenere il codice sorgente della tua versione attraverso un server in rete gratuitamente [...]

Anche la AGPLv3 è compatibile con la GPLv3.

4.5.5 BSD

BSD, come si intuisce dall'acronimo che sta per **Berkeley Software Distribution**, è una famiglia di licenze che nasce in ambiente accademico; le licenze BSD sono, in senso lato, le più libere di tutte quelle viste finora perché non contengono l'effetto virale caratteristico delle licenze copyleft. Il software rilasciato sotto BSD può essere modificato e ridistribuito scegliendo la licenza che più si preferisce; questo significa che un software BSD può essere modificato e reso un software chiuso.

MacOS X di Apple si basa proprio su un precedente sistema operativo UNIX-like molto conosciuto e usato, chiamato FreeBSD, modificato e poi reso un software proprietario a pagamento.
La stessa cosa ha fatto Microsoft adottando lo stack di rete di un altro sistema operativo UNIX-like rilasciato sempre sotto licenza BSD per i propri

prodotti.

Senza addentrarci troppo nei dettagli storici che hanno portato alla nascita di tale licenza, basti sapere che tale licenza è nata per gestire progetti finanziati con soldi pubblici; il principio di fondo è stato di permettere a chiunque di utilizzare i risultati da progetti con investimenti pubblici, così da stimolare l'innovazione e attività commerciali conseguenti.
Questa terza via imboccata dalla licenza BSD, rimanendo in linea con il gioco di parole adottato per copyleft per contrapporsi al copyright, è chiamata anche copycenter.
L'unico obbligo che portava licenza, cioè di citare gli autori, è stato rimosso con l'introduzione della cosiddetta licenza BSD modificata nel '99.

4.5.6 Apache License 2.0

La Apache License, inizialmente scritta dalla Apache Software Foundation per il celeberrimo Apache Web Server, è un'altra licenza che ha riscosso molto successo nel mondo FLOSS ed è diventata la scelta predefinita di migliaia di progetti Open Source.
Come altre licenza storiche anche questa licenza si è evoluta nel tempo, cambiando forma e nome, per diventare ora la Apache License 2.0 qui in esame.
Come per le licenze BSD anche la Apache License permette di utilizzare il software, modificarlo e rilasciarlo con la licenza che più si preferisce; come per la BSD, un software rilasciato sotto Apache License può essere trasformato in un software proprietario.
Per ridistribuire il codice l'unico vincolo imposto è di indicare chiaramente che si è utilizzato anche codice con licenza Apache ponendo in allegato al software un nota che cita il codice utilizzato e gli autori di tale codice.

Visto che di fatto è possibile scegliere qualsiasi licenza per ridistribuire il

codice la Apache License è compatibile con GPLv3; questo significa che è possibile mescolare software GPL e Apache nello stesso programma rilasciando il prodotto finale sotto GPL.

4.5.7 Microsoft Shared Source Initiative

Avviata a metà del 2007, la Microsoft Shared Source Initiative è nata con l'intento di permettere a soggetti terzi di aver un qualche tipo di accesso al codice sorgente delle applicazioni Microsoft in modo da favorire il lavoro di sviluppatore e governi che desiderino analizzarlo.
Nell'ambito di questo progetto Microsoft ha rilasciato cinque diverse licenze di cui due, la Microsoft Public License**,** abbreviata come Ms-PL, e la Microsoft Reciprocal License, abbreviata come Ms-RL sono state accettate dall'Open Source Initiative.
.
La **Microsoft Public License** è la più permissiva delle due e permette di ridistribuire sia il codice sorgente che il binario, per scopi commerciali o meno.
Se si ridistribuisce il codice sorgente è obbligatorio usare la stessa licenza, per la forma binaria invece si può usare un'altra licenza purché compatibile.
Siccome la licenza non introduce particolari novità la Free Software Foundation consiglia di utilizzare al suo posto la Apache License per evitare il problema della proliferazione delle licenze (vedi paragrafi precedenti).

La **Microsoft Reciprocal License** risulta concettualmente molto simile alla LGPL e permette l'utilizzo e la ridistribuzione del codice purché si mantenga la licenza.
Come per il software LGPL, codice sotto Ms-RL può essere integrato all'interno di altro software rilasciato con una licenza differente, purché per la parte sotto Microsoft Reciprocal License sia sempre disponibile la copia della licenza e del codice sorgente.

Come accennato esistono altre tre licenze che fanno parte della Shared Source Initiative ma che tuttavia NON sono licenze Open Source:
la Microsoft Reference Source License (Ms-RSL) permette esclusivamente di visionare il codice sorgente ma non di modificarlo o tanto meno di ridistribuirlo;
la Microsoft Limited Public License (Ms-LPL) viola la norma sulla neutralità tecnologica della Open Source Definition in quanto restringe il campo di utilizzo del codice sorgente a sistemi Windows;
come la precedente la Microsoft Limited Reciprocal License (Ms-LRL) viola la Open Source Definition perché non tecnologicamente neutra.

4.5.8 Tabella Riassuntiva

Licenza	Link da codice con licenza differente	Rilascio delle modifiche sotto un'altra licenza
GPLv2	no	no
GPLv3	no	no
LGPLv3	sì	no
Affero	no	no
BSD	sì	sì
Apache License 2.0	sì	sì
Microsoft Public License	no	no
Microsoft Reciprocal License	sì	no

4.6 Affermazione dell'Open Source

Il Free Software prima e l'Open Source poi sono sempre stati fin dall'inizio molto popolari all'interno delle comunità di sviluppatori e in ambito accademico, altrettanto non si può dire però per il mondo economico, in cui lo scetticismo ha sempre prevalso su gli effettivi vantaggi di questo nuovo

modello.
La fase embrionale del Free Software e dell'Open Source si considera idealmente di solito durare fino al 1998, quando la rivista Fortune dedicò la copertina a Linus Torvalds, creatore di Linux, e grandi aziende come Oracle fecero il porting delle loro applicazioni sotto Linux; ciò decreto la maturità di Linux e dell'intero movimento anche nel mondo economico.
Tuttavia ancora oggi ci troviamo in una fase transitoria: se da un lato infatti numerose sono le aziende già entrate nel mercato dell'Open Source, come IBM, Oracle, SUN e Novell, ci sono anche aziende come Microsoft che prima l'hanno fortemente osteggiato e solo recentemente hanno incominciato a sfruttarlo (vedi Shared Source Initiative nei paragrafi precedenti).

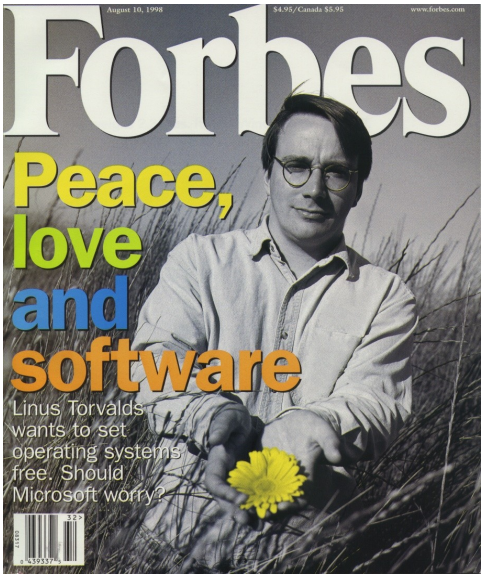


Figura 23: Copertina di Forbes dedicata a Linus Torvalds e all'Open Source

Un caso emblematico, in negativo, di aziende passate al modello Open Source è quello di Netscape; nel 1998 il codice sorgente del browser Netscape Comunicator venne reso Open Source per competere con Microsoft che distribuiva il suo browser gratuitamente e direttamente dentro al sistema operativo (da cui le prime condanne per violazione delle regole sulla

concorrenza).
Con questa scelta Netscape ha reso ufficiale l'importanza dell'Open Source all'interno delle strategie di mercato di una azienda.
Benché il rilascio sia avvenuto a ritmi serrati, Netscape non è riuscita ad avvantaggiarsi di questa svolta (forse perché avvenuta troppo tardi); l'avanzata di Internet Explorer è stata troppo veloce e la quota detenuta da Netscape Communicator è stata fagocitata. Tuttavia è fondamentale ricordare che dal codice sorgente di Comunicator è nato Mozilla, ora Firefox, il secondo browser più utilizzato al mondo e unico vero concorrente di Internet Explorer.
Questo esempio è emblematico sulla capacità dei progetti Open Source di mantenersi e propagarsi anche oltre la sopravvivenza di chi originariamente ha creato il progetto.
Firefox sta pian piano conquistando una fetta di utenti sempre maggiore e sta riuscendo dove nessun altro finora aveva avuto successo; ancora una volta questo dimostra come in un segmento molto competitivo e chiuso l'Open Source riesca a fare breccia perché non segue le regole consolidate del software proprietario.
Nessuno si può oggi permettere di combattere alla pari con Microsoft e Internet Explorer o Windows; Firefox e Linux stanno avendo un discreto successo perché utilizzano un approccio diverso al mercato, riuscendo ad abbassare drasticamente i costi di sviluppo e ad aumentare la qualità del software.

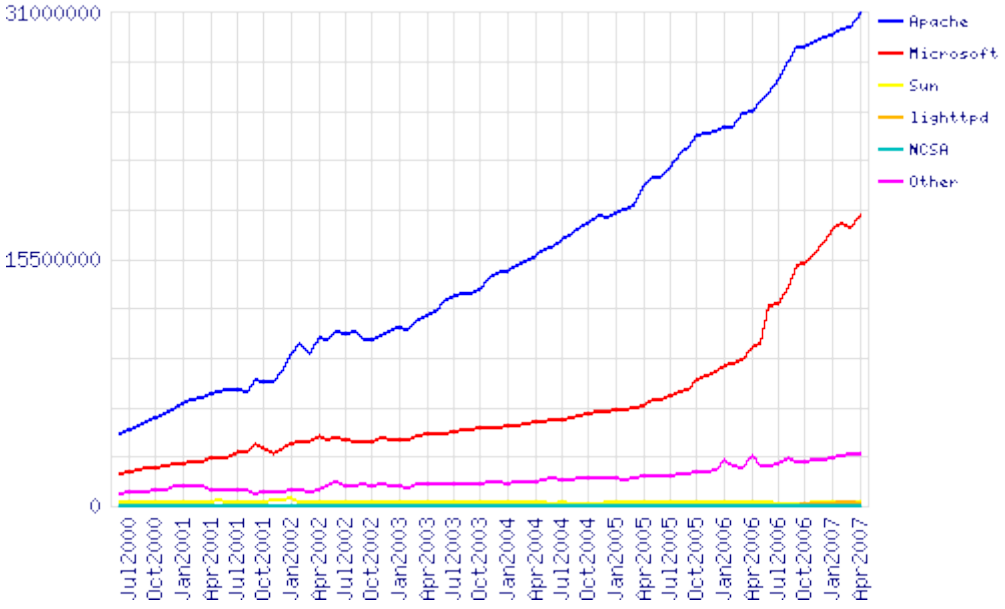


Figura 24: Numero di web server attivi

Uno dei grandi prodotti di successo dell'Open Source è senza dubbio l'Apache Web Server che detiene un record di presenza su Internet, gestendo oltre la metà di tutti i siti Internet del mondo, per un totale di circa 80 milioni di siti internet serviti.

Per fare alcuni nomi di utilizzatori di Apache web server citiamo aziende e organizzazioni di importanza internazionale tra le quali: amazon.com, americanexpress.com, whitehouse.gov, sony.com, cisco.com e hp.com, e locali come: cineca.it, garr.it o libero.it.
Altro successo enorme che arriva da Internet è Bind che rappresenta il 77% di tutti i server DNS di Internet.
Come si vede l'Open Source annovera prodotti di altissimo profilo, a cui è richiesta elevata affidabilità e sicurezza, nonché clienti di altissimo profilo.
Il più famoso prodotto Open Source rimane comunque Linux per cui è necessario spendere qualche parola in più:
come già accennato per il progetto GNU di Richard Stallman furono sviluppati molti dei programmi chiave necessari per creare un sistema operativo ma nel 1991, sette anni dopo l'inizio del progetto, mancava ancora un mattone il pilastro che reggesse tutto il sistema, il kernel.

Il buco venne colmato nel 1991 da Linux, un kernel creato dall altra parte del mondo, in Finlandia, da un studente dell università di Helsinki, di nome Linus Torvalds, che sviluppò per hobby nel giro di un anno un proprio kernel.
La prima versione del kernel Linux, la 0.1 venne rilasciata, come poi tutte le altre, sotto i termini della licenza GPL; ora, benché il kernel Linux fosse ancora in fase altamente sperimentale, il sistema GNU era completo, la GPL avevo reso possibile una cosa altrimenti impossibile: la collaborazione di soggetti diversi sparsi in tutto il mondo con obbiettivi differenti.
Alla fine del 1991 unendo il software sviluppato per il progetto GNU e il kernel Linux sviluppato da Torvalds si aveva a disposizione un sistema operativo completamente funzionante e completamente libero.

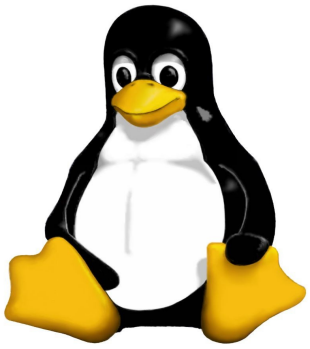


Figura 25: Tux, la mascot di Linux

Ovviamente anche la FSF aveva intenzione di scrivere un proprio kernel per completare il progetto GNU, ma Linux risultò essere molto più semplice e veloce da sviluppare del kernel GNU che aveva soluzioni tecniche all avanguardia ma molto difficili da programmare.
Ancora una volta la GPL aveva permesso che soluzioni differenti fossero sviluppate in contemporanea e aveva portato alla selezione di quella migliore, una sorta di darwinismo informatico.
Tuttora il kernel GNU, rilasciato sotto il nome di Hurd, è sperimentale e incompleto, mentre Linux è utilizzato su una moltitudine di piattaforme hardware, da sistemi embedded, come palmari e cellulari, ai PC fino ai super-calcolatori IBM e Silicon Graphics per il calcolo scientifico.

Attualmente il numero di utenti Linux è stimato in circa 29 milioni.

Il luogo comune che l'Open Source sia poco diffuso non trova riscontro nella realtà se non in un settore: i Desktop.
La stragrande maggioranza del software per il segmento Desktop è effettivamente al momento per circa l'80% made in Redmond.
Al di fuori però di questo ambito in cui Windows detiene un monopolio di fatto, il software open è molto diffuso e anzi talvolta,come nel caso di Bind, è l'unico vero soggetto in gioco.

Man mano che un software Open Source acquista consensi e si diffonde acquista anche sviluppatori e ciò allarga a sua volta la base dei tester rendendo il software migliore; questo porterà ad avere più utenti e sviluppatori innescando un circolo virtuoso.
Come già ampiamente descritto da Eric Raymond in La cattedrale e il Bazar l'Open Source offre un modello di sviluppo che favorisce la revisione paritaria del codice e, soprattutto, una base di testing e controllo molto ampia che difficilmente una software house di medie dimensioni si può permettere.

Per citare Raymond:

> *Given enough eyeballs, all bugs are shallow.*

5 Dinamiche dei progetti Open Source e il radicale cambiamento apportato nei processi aziendali

Normalmente in un progetto proprietario la struttura che sta dietro allo sviluppo di un software è rigida e ben definita; un piccolo e chiaramente circoscritto gruppo di persone pagate dall'azienda si concentrano nella scrittura del codice sorgente, il gruppo di management decide la roadmap del progetto, un altro gruppo si occuperà del marketing e della comunicazione, un altro del sito internet, un altro del supporto e così via; più l'azienda è grossa più queste figure saranno distinte e slegate tra loro mentre nelle aziende di ridotte dimensioni solitamente queste figure tendono a sovrapporsi ampiamente.
I processi aziendali a loro volta sono strutturati, esiste una gerarchia, un superiore di riferimento e così via; i mezzi di comunicazione sono formali e diretti.
Al contrario in un progetto open di solito vi è ristretto gruppo di persone che rappresentano l'autorità nel progetto, e non necessariamente sono legate all'azienda che ha iniziato il progetto, e una grosso volume di persone che entrano ed escono dal progetto dando piccoli contributi.
Il compito di chi gestisce un progetto open è proprio quello di tenere basse le barriere all'ingresso del progetto per favorire l'acceso di nuove persone.
Le relazioni sono paritarie e avvengono attraverso mezzi di comunicazione indiretti come l'email o il forum e si deve far fronte a differenze culturali tra persone geograficamente distanti.
Lo sforzo maggiore richiesto a chi si avvicina per la prima volta ad un progetto Open Source è proprio di cambiare radicalmente ottica, passando da un modello di sviluppo che potremmo chiamare privato, ad uno pubblico, davanti agli occhi di tutti; quello che a prima vista può sembrare caotico in realtà segue semplicemente leggi diverse, dove l'autorevolezza che una persona riesce ad esprime è determinante più che l'autorevolezza che si detiene come posizione in azienda.

Come spiegato nel paragrafo sul Free Riding a nessuno piace sentirsi derubato del proprio lavoro ed è perciò di vitale importanza essere completamente onesti con la comunità, scelte ambigue o poco chiare genereranno inevitabilmente malcontento e daranno cattiva fama al progetto; questo vuol dire innanzitutto evitare discussioni private o prendere decisioni unilateralmente, facendo ovviamente i dovuti distinguo.
Questo può spaventare perché in parte vuol dire perdere il controllo su alcuni aspetti che prima erano di esclusiva pertinenza dei creatori del progetto; i partecipanti della community diventano un soggetto con cui dialogare e di cui tenere conto, ma questo è proprio ciò che ci si aspetta dalla community!
Il risvolto positivo di tutto questo è che il prodotto sarà sempre vicino alle aspettative dei clienti.

Lo sviluppo del codice diventa poi inaspettatamente un nuovo punto di contatto con il pubblico; se prima infatti il codice rimaneva segreto, ed eventualmente l'utente poteva solo constatare un crash del software, adesso la qualità del codice è sotto gli occhi di tutti e può contribuire a creare una buona o cattiva reputazione del progetto.
Ma non solo, il processo di sviluppo e i programmatori stessi, prima totalmente sconosciuti all'esterno dell'azienda, entrano in contatto direttamente con la comunità; se si rende open un progetto proprietario precedente i programmatori in particolare potrebbero trovarsi a disagio a doversi confrontare con soggetti esterni che analizzano il proprio lavoro. E' importante quindi rendersi conto fin da subito della situazione e dello stato del codice e prepararsi.
Anche il modello dei rilasci del software viene cambiato: il modello classico è organizzato come una sorta di scala con bruschi cambiamenti in concomitanza dei rilasci della major release.
Una release maggiore è seguita da un ciclo ben scadenzato nel tempo di patch per arrivare quindi a una nuova release minore di mantenimento che sintetizza grosso modo tutte le patch rilasciate fino a quel momento.

Il rilascio della major release è solitamente dettato da esigenze commerciali e quando si sforano i tempi si ha la tendenza a rilasciare il software anche se effettivamente non completo, scaricando lo sviluppo non ancora completato nelle release minori.
Al contrario nel mondo open c'è un susseguirsi di rilasci minori, molto fitti nel tempo se paragonati al caso precedente, con una crescita del progetto costante e graduale, e il software tendenzialmente è rilasciato solo quando effettivamente pronto.

Kernel Version	Release Date	Days of Development
2.6.11	2005-03-02	69
2.6.12	2005-05-17	108
2.6.13	2005-08-28	73
2.6.14	2005-10-27	61
2.6.15	2006-01-02	68
2.6.16	2006-03-19	77
2.6.17	2006-06-17	91
2.6.18	2006-09-19	95
2.6.19	2006-11-29	72
2.6.20	2007-02-04	68
2.6.21	2007-04-21	81
2.6.22	2007-07-08	75
2.6.23	2007-10-09	94
2.6.24	2008-01-24	108

Figura 26: Date di rilascio del branch 2.6.x del kernel Linux

Dalla tabella riportata qui sopra si può vedere come i rilasci siano molto frequenti e il ciclo di sviluppo massimo è intorno ai tre mesi.

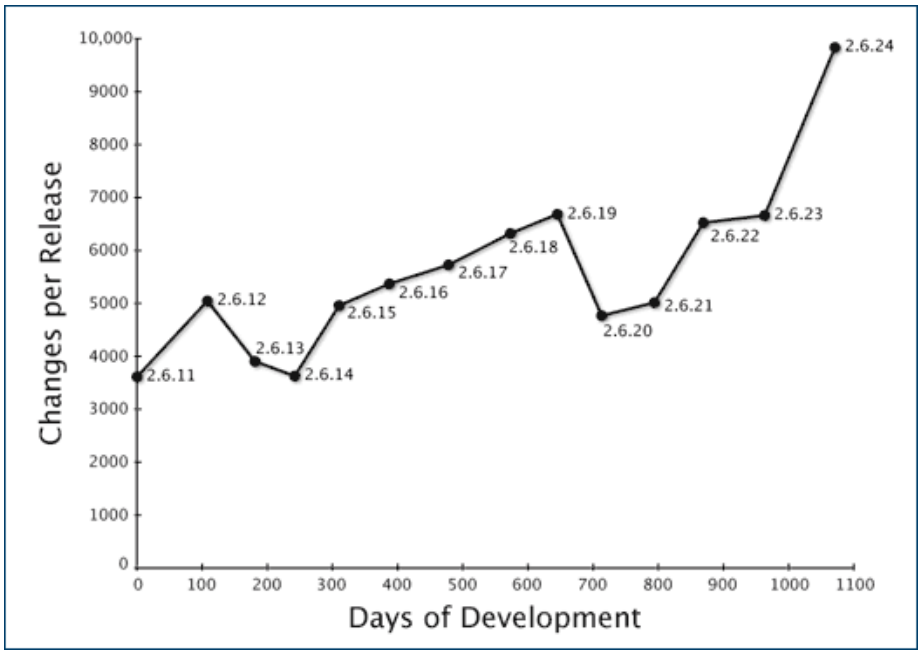


Figura 27: Cambiamenti apportati nel kernel Linux rispetto ai giorni di sviluppo

Dal grafico riportato qui sotto è ancora più evidente come vengano abbandonato uno schema organizzativo rigido dei rilasci.

Per capire quanto sia diverso l'approccio basta considerare il caso Wine, un progetto nato per implementare su Linux le API di Windows, una sorta di layer di compatibilità tra i due sistemi operativi che permette di far girare gli applicativi Windows su Linux; questo progetto, di enorme successo, è attivo da 15 anni ma la versione 1.0 verrà rilasciata solo a metà del 2008, benché i rilasci siano piuttosto frequenti.
Questo è ovviamente un caso limite ma serve per esplicare come il progetto sia considerato come un continuum nel tempo e non come una serie precisa e definita serie di release, relativizzando di molto l'importanza al numero di release.

In apparentemente contrasto con questo metodo di procedere community based risultano le leggi di Brooks, premio Turing nel '99 e autore di The Mythical Man-Month :

"Adding manpower to a late software project makes it later"

Il problema è di origine principalmente organizzativa: se infatti nello sviluppo sono impiegate poche persone il contatto e lo scambio di idee tra esse può essere informale e avvenire ad esempio direttamente a voce; questo ovviamente non è possibile anche solo le persone coinvolte siano qualche decina.
La questione è valida sia per il software proprietario quanto per quello Open Source ma il punto di svolta è la tecnologia oggi a disposizione che consente di portare l'organizzazione ad un nuovo livello. Come già ripetuto iniziare un progetto Open Source è più faticoso che iniziare un progetto proprietario perché richiede di costruire tutta quella infrastruttura necessaria alla comunità come un sito internet, un blog, un wiki e così via.
Effettivamente questi sono tutti strumenti che abilitano la comunicazione e lo sviluppo condiviso a distanza e servono appunto per aggirare, o quanto meno mitigare, i problemi sottolineati dal teorema di Brook, permettendo di automatizzare tutti i processi e le regolamentazioni necessarie all'organizzazione. E' dimostrato dal successo di parecchi progetti open che quando viene raggiunta una certa dimensione il progetto è in grado non solo di auto-mantenersi ma anche di prosperare; il problema vero è capire quale sia la giusta dimensione, cosa certo non facile.
In conclusione lo sforzo iniziale è ripagato dalla possibilità di gestire una complessità crescente in maniera quadratica con uno sforzo che invece procede linearmente.

Un'altra legge apparentemente in contrasto con il metodo Open Source è:

> *The bearing of a child takes nine months, no matter how many women are assigned. Many software tasks have this characteristic because of the sequential nature of debugging.*

Banalizzando, ci sono attività intrinsecamente non parallelizzabili e il fatto che siano coinvolte molte persone non porta alcun vantaggio; il che è tautologico e assolutamente non in conflitto con il modello Open Source.

E' ovvio che non c'è un modo per impiegare nove donne per far partorire un bambino in un mese, ma il problema è mal posto: il fatto che il debugging sia sequenziale è immutabile, non è possibile prendere 100 righe di codice e farle debuggare a 10 programmatori in contemporanea con ciascun programmatore che controlla solo 10 righe.
Il debugging in realtà è un'attività molto aleatoria e soggetta al caso, l'autore originario del codice probabilmente avrà una maggiore probabilità di scoprire un bug, ma rimane comunque un ampio margine di incertezza. Avere a disposizione più persone che controllano lo stesso codice aumenta drasticamente le possibilità che un bug venga individuato aumentando la qualità del codice nel tempo ad un tasso maggiore di quello che si avrebbe con meno personale; questo garantisce un tempo di sviluppo più corto e maggiore qualità.

Un progetto Open Source è fondato sulle relazioni umane dei diversi soggetti che partecipano alla community e ciò porta con se una serie di oneri aggiuntivi rispetto ai progetti chiusi, compensati però dai considerevoli vantaggi visti fin qui.
Per poter gestire efficacemente una community bisogna comprendere le motivazione delle persone che vi partecipano nonché il tipo di persone che vi partecipano.

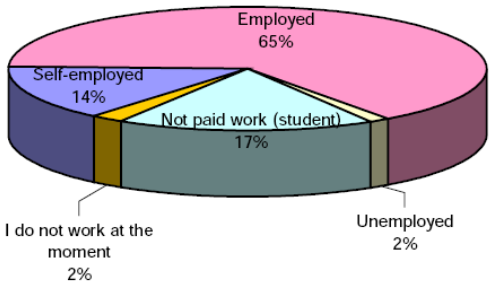


Figura 29: Condizione lavorativa dei developer di software Open Source

Chi partecipa ai progetti Open Source solitamente sono professionisti e non semplici smanettoni che scrivono codice per hobby; le figura riportate

sopra e sotto sono frutto di una ricerca del 2002 finanziata dalla comunità Europea e portata a termine dall'Università di Maastricht, in cui è evidenziato come circa l'80% degli sviluppatori sia occupato e solo il 17% siano studenti. Inoltre il 67% degli sviluppatori sono professionisti del settore IT, in particolare il gruppo più folto è quello degli ingegneri del software seguito da un 16% di studenti, che sottolinea come l'Open Source sia comunque molto diffuso nell'ambito accademico e di ricerca.

Company Name	# of Changes	% of Total
None	11,594	13.9%
Unknown	10,803	12.9%
Red Hat	9,351	11.2%
Novell	7,385	8.9%
IBM	6,952	8.3%
Intel	3,388	4.1%
Linux Foundation	2,160	2.6%
Consultant	2,055	2.5%
SGI	1,649	2.0%
MIPS Technologies	1,341	1.6%
Oracle	1,122	1.3%
MontaVista	1,010	1.2%
Google	965	1.1%
Linutronix	817	1.0%
HP	765	0.9%
NetApp	764	0.9%
SWsoft	762	0.9%
Renesas Technology	759	0.9%
Freescale	730	0.9%
Astaro	715	0.9%
Academia	656	0.8%
Cisco	442	0.5%
Simtec	437	0.5%
Linux Networx	434	0.5%
QLogic	398	0.5%
Fujitsu	389	0.5%
Broadcom	385	0.5%
Analog Devices	358	0.4%
Mandriva	329	0.4%
Mellanox	294	0.4%
Snapgear	285	0.3%

Figura 30: Partecipazione allo sviluppo del kernel Linux

Nella figura sottostante, per fare un esempio concreto, si può vedere la percentuale di contributi offerti al kernel Linux, per farsi un idea di chi effettivamente sviluppa software Open Source:Come si vede il 14% delle modifiche non arriva da professionisti di qualche azienda mentre quasi il 30% arriva dalle tre aziende leader nel settore: Red Hat, Novell e IBM.

Le motivazioni che spingono a partecipare a una community sono le più disparate ma vedendo anche il tipo di persone che partecipano alle community è f

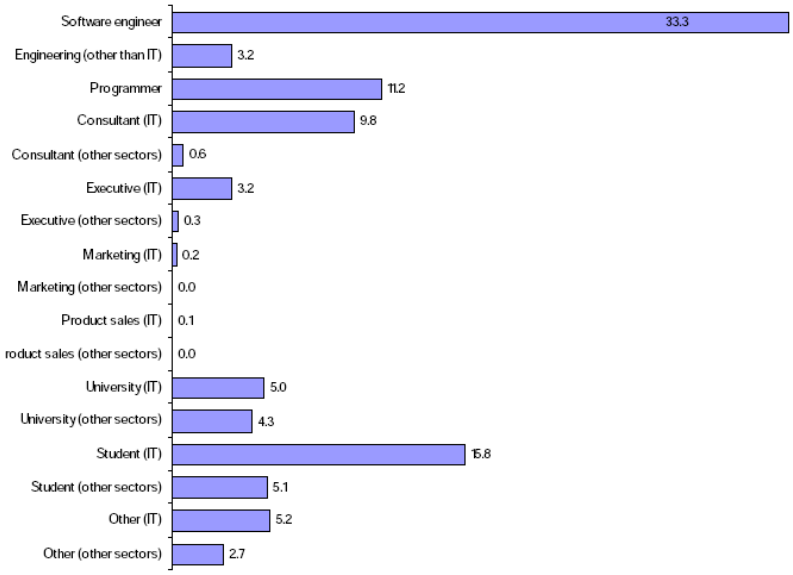


Figura 31: Statistiche sull'impiego degli sviluppatori open

Benché una persona non partecipi attivamente allo sviluppo, non scriva cioè del codice, esistono diversi livelli di coinvolgimento e diverse attività di cui può occuparsi; il supporto agli altri utenti o la documentazione sono ad esempio due campi in cui anche utenti non tecnici possono cimentarsi.
Tra chi arriva a scrivere del codice vengono solitamente identificate tre tipologie di approccio a seconda del livello di competenza tecnica e interesse verso il progetto: da un lato si può trovare il semplice utente che si trasforma in beta tester in seguito alla necessità di correggere un bug; oppure l'utente professionale che necessità una particolare feature o vuole cambiare il funzionamento di un aspetto per esigenze personali; infine lo

studente/ricercatore che utilizza il software open già disponibile come base di partenza per esplorare nuovi campi di ricerca.
Queste tre possibilità di interazione con il progetto sono il punto forte del software Open Source e non trovano riscontro nella quasi totalità dei casi per il software proprietario.
Un altro vantaggio che si ha lavorando a stretto contatto con la community è di riuscire ad identificare nuovo personale da assumere quando ce ne sia l'esigenza; la comunità diventa un bacino dove testare e selezionare il personale nonché uno strumento per la formazione interna.

6 Il modello economico dell'Open Source

Ancora oggi è facile trovare una certa diffidenza verso il modello Open Source, in particolar modo da persone provenienti dal mondo dell'IT.
Il modello classico, a cui il mercato informatico è abituato, è quello che ci portiamo dietro dagli anni '80 e che ha fatto nascere grandi colossi come IBM, Microsoft o Oracle; queste aziende hanno generato fatturati da miliardi di dollari vendendo principalmente licenze cioè dando ai clienti il proprio software in utilizzo in cambio di una certa somma di denaro.
La questione che viene solitamente sollevata è come si possa replicare il successo di Microsoft nel mondo Open Source; la risposta è: non si può.
La domanda in realtà è fuorviante perché Microsoft è il baluardo del modello classico, quello dell'Open Source è un modello totalmente nuovo per un altro scenario, diverso da quello in cui prospera BigM.

La domanda corretta da porsi è quanto sia profittevole il mercato dell'Open Source e non come sia prono a replicare il modello Microsoft.
Venendo dal modello chiuso viene da chiedersi come possano quindi sopravvivere queste aziende regalando il software, togliendosi di fatto la fetta maggiore dei loro introiti.
Eppure IBM è oggi una delle maggiori aziende impegnate nell'Open Source, Oracle ha una propria distribuzione Linux (Unbreakable Linux). Microsoft possiede il più grande laboratorio di sperimentazione di prodotti Open Source al mondo.
Nuove aziende sono nate che prosperano sull'Open Source come Red Hat, aziende storiche in difficoltà si sono reinventate nell'Open Source, come Novell.
Citando le parole di Robert Young, CEO di Red Hat:

> *La maggior parte delle imprese software, siano basate su software libero o proprietario, falliscono. Dato che fino a tempi recentissimi tutte le imprese software erano del tipo proprietario binary-only, è prudente affermare che col modello Intellectual*

Property di sviluppo e marketing è difficile guadagnarsi da vivere. Lo era anche, naturalmente, cercare l'oro col setaccio durante la corsa all'oro del XIX secolo. Ma quando le società software trovano la vena generano una gran quantità di denaro, proprio come nelle antiche corse all'oro; molti, dunque, decidono di assumersi rischi per avere l'opportunità di trovare la vena anche loro.

Nessuno si aspetta che far soldi col software libero sia cosa facile. E' bensì una sfida, ma non più grande che con il software proprietario. Di fatto, con il software libero il guadagno si genera esattamente nello stesso modo che con quello proprietario: creando un grande prodotto, commercializzandolo con accortezza e fantasia , prendendosi cura dei clienti e, di lì, costruendo un grande marchio che sia sinonimo di qualità e di servizio alla clientela.

Nei paragrafi successivi verranno analizzate le motivazioni che rendono questo modello basato sul regalo così attraente sia da parte delle aziende che dei clienti e si tenterà di rispondere alle numerose domande che permangono.

'The thing I'm puzzled by is how there will be a software industry if there's open-source,' Jim Gray

6.1 L'Open Source per il cliente

Per poter capire perché l'Open Source sta avendo così successo è importante prima comprendere quali sono i vantaggi per il cliente e perché il cliente ha tutto l'interesse ad acquisire software open a discapito di quello proprietario (chiamato anche privativo).
La revisione paritaria del codice, nonché l'intera struttura dello sviluppo open vista nei capitoli precedenti, sono uno strumento formidabile per ottenere software di elevata qualità e affidabilità ad un costo minore; oltre

all'aspetto sicurezza, molto importante, questo si traduce per il cliente anche in minori blocchi, meno interruzioni del lavoro, meno assistenza richiesta e quindi in definitiva contribuisce ad avere un minore TCO (total cost of ownership).
La disponibilità del codice sorgente consente inoltre una personalizzazione e una raffinazione del prodotto estrema, permettendo di modificarne ogni aspetto e poterlo adattare alle proprie esigenze specifiche.
Aver a disposizione il codice vuol dire anche open format: molti vendor di software proprietario utilizzano standard chiusi come mezzo per legare il cliente alle proprie soluzione e rendere difficoltoso se non impossibile cambiare vendor.
Alcuni formati come quelli usati da AutoCad o da Microsoft Office sono formati chiusi, proprietari, segreti, e il cliente dipende completamente dalla casa madre; questo comporta non solo una incompatibilità verso l'esterno o verso altri software, ma anche interna, in quanto è sempre la casa madre a decidere quando interrompere il supporto a tale formato. Essendo il modello economico del software tradizionale basato sulla vendita della licenza lo scopo principale dei vendor è appunto quello di massimizzare la vendita delle licenze, la gestione dei formati è uno degli strumenti che i vendor hanno per imporre al cliente un aggiornamento.
La questione dei formati chiusi apre di conseguenza anche quella dell'interoperabilità; i formati chiusi diventano nuovamente uno strumento per forzare il cliente verso soluzioni dello stesso vendor che promettono di integrarsi con facilità.
Quello degli open standard è un argomento particolarmente delicato per le pubbliche amministrazioni che hanno il dovere di mantenere i dati dei cittadini; se prendiamo un esempio classico, come quello dell'anagrafe, la scelta di un formato proprietario per i dati, come potrebbe essere quello di Word, è una scelta a dir poco rischiosa.
La situazione potenzialmente esplosiva che si prefigura è di trovarsi tra qualche decina d'anni con immensi archivi di dati non leggibili.

L'Open Source porta con se altri due vantaggi oltre a quelli visti finora: il primo è di svincolarsi dal single vendor lock-in, il secondo è di mettersi al riparo da un'eventuale fallimento o indisponibilità della casa madre.
La questione dei formati è immediatamente risolto in quanto la costruzione del formato dei dati è contenuto nel codice stesso.
Inoltre nel momento in cui la casa madre non dovesse essere più disponibile o in grado di far fronte agli impegni, o per un qualunque motivo il cliente volesse cambiare fornitore, la disponibilità del codice garantisce la continuità del progetto oltre la vita stessa del vendor.
Il caso di fallimento del proprio fornitore può risultare un evento particolarmente catastrofico nel caso di software chiuso; dopo un lasso di tempo variabile il cliente si trova costretto a cambiare software in cambio di uno che venga aggiornato e per cui venga offerta assistenza.
Nel caso di software open il cliente può sempre pagare un'altra software perché se ne occupi e continui lo sviluppo (si veda il caso Netscape nei capitoli precedenti).

E' evidente come per il cliente l'Open Source, anche a parità di costo iniziale, sia la strada più conveniente e affidabile.
I due scogli più grandi per la diffusione dell'Open Source sono due, uno di carattere comunicativo e l'altro pratico: al di fuori del mondo informatico non è chiaro cosa sia l'Open Source e come funzioni la sua economia; il fatto poi che il software possa essere utilizzato gratuitamente crea immediatamente diffidenza risultando al primo impatto una scelta controproducente.
Questo aspetto dovrebbe risolversi da solo con il tempo in seguito alla diffusione dell'Open Source e alla crescita informativa dei clienti.
Il problema pratico è invece legato all'assistenza: essendo Apache il più importante dei server web, trovare un supporto adeguato alle proprie esigenze è abbastanza facile per non dire scontato; nel mercato Desktop dove l'80% del software attualmente installato è Microsoft trovare il supporto adatto può essere più difficoltoso.

Il problema è come al solito ricorsivo: non c'è offerta se non c'è domanda, ma d'altra parte non c'è domanda perché non c'è offerta; chi voglia passare a Linux avrà paura di non trovare supporto e rimarrà fermo, d'altra parte le aziende non forniranno supporto perché non c'è richiesta.

6.2 L'Open Source per il produttore

Vediamo ora cosa offre l'Open Source visto dalla parte di chi sviluppa software, iniziando dai casi in cui la svolta open può risultare scorretta.
Come ogni cosa, l'Open Source non è una panacea per ogni male, ed esistono dei casi in cui banalmente non è la strada migliore.
Innanzitutto è necessario dividere le aziende che fanno del software il proprio core business dal resto.
Le aziende infatti che non si occupano di software, ma che per un motivo o per l'altro si trovano a svilupparne per le proprie esigenze interne, hanno l'interesse primario di portarsi in casa del software poco costoso, solido, testato e molto diffuso; sono evidentemente le candidate perfette per l'acquisizione di software Open Source, indipendentemente dal tipo di software di cui si parli.
Le aziende, al contrario, che di informatica ci campano, hanno l'esigenza di massimizzare i profitti ed è fondamentale per fare questo analizzare il mercato nel suo complesso.
Il caso più ovvio in cui il modello Open Source è totalmente perdente, dal punto di vista di chi fa il software, è quello di un mercato poco concorrenziale; tutte le volte che ci troviamo di fronte ad una condizione di dominio tecnologico in un settore con pochi concorrenti o palesemente di livello inferiore, adottare una strategia Open Source risulta meno efficiente, per non dire disastrosa.
Se prodotti come ad esempio Fluent o AutoCad, che sono degli standard de facto nella loro nicchia e non hanno concorrenti di rilievo, venissero resi aperti, l'intero segmento verrebbe immediatamente bruciato , senza ricavare

particolari vantaggi dal nuovo modello open.
In particolare AutoCad, che non ha la complessità di Fluent e che non richiede particolare supporto , subirebbe le perdite più pesanti.
Altro punto da valutare sono le cosiddette caratteristiche differenzianti del prodotto; la definizione è sfuggevole, ad ogni modo tutto quello rende il proprio prodotto diverso dagli altri, quello che riesce ad offrire in più, è una caratteristica differenziante.
Facendo un esempio puramente immaginario, se avessi un IDE per lo sviluppo in C che introduce come novità assoluta l'auto completamento del codice, il mio prodotto ha una caratteristica unica che nessun altro prodotto possiede: quella è una caratteristica differenziante.
Condividere con i concorrenti quella caratteristica sviluppata internamente sostenendo completamente i costi di ricerca e sviluppo sarebbe semplicemente stupido.
In linea di principio risulta vantaggioso per tutti cercare di dividere i costi e i rischi per lo sviluppo di tutte le caratteristiche non differenzianti, e tenersi gelosamente strette quelle che invece lo sono.
Il software è un bene immateriale, una volta creato l'originale il costo di replica è praticamente nullo; questo ha ovviamente un grosso peso, se pensiamo ad altre industrie, come quell'automobile, l'idea di regalare la macchina per vendere l'assistenza suona un po' assurdo.
La possibilità di fare copie e di poter riutilizzare il codice facilmente è un'altra caratteristica unica del software; una portiera va in una sola macchina, ma non è così per il software.
L'Open Source consente quindi di diminuire i costi iniziali di sviluppo dividendoli tra più soggetti e di ridurne i tempi, consentendo un time to market minore.

Se il software è gratuito vediamo le fonti di guadagno sono necessariamente altre:

- Vendita di software accessorio
- Consulenza

- Assistenza alla prima installazione
- Assistenza di mantenimento
- Sviluppo di nuovo codice
- Integrazione
- Aggiornamenti
- Certificazioni di sistema
- Formazione
- Documentazione
- Personalizzazione
- Hardware
- Comodità
- Doppia licenza
- Gestione del marchio

La via intermedia tra l'Open Source e il software proprietario è di offrire delle soluzioni di base completamente open e software accessorio a pagamento; in questo caso ricade ad esempio Novell, che offre a pagamento alcuni software molto specifici indirizzati al mercato enterprise.

La consulenza non cambia tra software open e proprietario, quello che però si può offrire in più al cliente è la varietà e la possibilità di testare per intero il software prima di adottarlo.

Nel caso di software complesso o di deployment ampi una grossa fetta di guadagni, sia per il software open che close, viene dalla cosiddetta assistenza alla prima installazione; il cliente paga per avere il supporto e le competenze tecniche per installare e configurare il sistema. Un ambito in cui questa voce ha un notevole peso è ad esempio negli ERP, dove l'assistenza e la personalizzazione la fanno da padrone. SAP potrebbe essere un buon candido per passare al modello Open Source, se non fosse in una situazione di predominio così evidente.
I vari concorrenti di SAP potrebbe invece trarre un grande vantaggio

dall'Open Source per concorrere e cercare di erodere quote ai concorrenti. Quanto detto non ha senso per software di uso comune o molto semplice; pensare di prosperare offrendo assistenza prima installazione ad esempio per un editor di testo potrebbe essere un problema.

L'assistenza di mantenimento è un aspetto molto importante per le aziende e che tende ad acquisire maggior peso con l'aumentare delle dimensioni dell'azienda; quello che le aziende cercano è la sicurezza, la certezza di poter chiamare qualcuno nel caso qualcosa non funzioni. Anche in questo caso non c'è alcuna differenza se si tratta di software libero o proprietario.

Lo sviluppo di nuovo codice è una realtà per le piccole aziende e le piccole software house dove si sviluppano piccole soluzioni ad hoc per problemi molto circoscritti e specifici; il vantaggio competitivo che offre l'Open Source è quello di poter sviluppare a costi minori riutilizzando codice già disponibile e di diminuire quindi i tempi di consegna e di poter offrire al cliente una applicazione a valore aggiunto con tutti i vantaggi per il cliente visti prima.

L'integrazione di sistemi è un'altra fonte di guadagno; l'integrazione è il valore aggiunto di per se, il fatto che il software sia open o meno non influenza il risultato. In particolare meno costa il software maggiore sarà l'offerta che si può proporre ai clienti e più tendenzialmente se ne utilizzerà facendo aumentare la richiesta di integrazione.

La scelta di far pagare gli aggiornamenti va valutata in base al tipo di clientela e di segmento che si considera; in alcuni ambiti, come quello della sicurezza, un software non aggiornato è inutile, quindi il vero valore non sta tanto nel software in se quanto negli aggiornamenti. Per fare un esempio concreto, un proxy che offra anche la possibilità di filtrare siti web attraverso black list è inutile se non si mantengono le liste aggiornate; una fonte di guadagno considerevole potrebbe essere offrire a pagamento quegli

aggiornamenti.
Un altra strada potrebbe essere di offrire a pagamento invece le patch o le nuove versioni in anteprima; questa è la strada intrapresa qualche tempo fa dalla distribuzione Linux Mandriva che avea creato una sorta di club, in cui si entrava pagando, che offriva appunto servizi privilegiati.

La certificazione di sistema è un altro tema molto importante in ambiente enterprise dove è richiesto che qualcuno fornisca la garanzia che un determinato software funzioni su un determinato hardware o in collaborazione con un altro software.
Questo aspetto in particolare è molto importante nell'ambito dei centri di calcolo o di grosse installazioni.

La formazione è un'esigenza generata da tutto il software; il vantaggio competitivo di chi offre software Open Source e di poter spostare l'attenzione dalla licenza alla formazione e vendere all'azienda non più un pezzo di carta ma nuovo know how da far entrare in azienda; acquistare nuove competenze per i propri dipendenti invece che una licenza è molto più allettante.

La documentazione è una fonte a volte trascurata ma può essere una voce interessante, in particolare se si considera in accoppiata con la formazione; grandi società come Novell offrono a pagamento non solo il classico manuale d'uso per il sistema operativo e simili, ma anche self study kit per le certificazioni professionali che offre. Oltre che attraverso i corsi è possibile quindi prepararsi ad affrontare le certificazioni anche autonomamente attraverso questi kit.

Quello in cui ovviamente il software Open Source eccelle è la flessibilità ed è relativamente facile e immediato offrire soluzione cucine sul cliente come servizio a pagamento.
In particolare più il software è complesso e articolato maggiore sarà questa

possibilità quindi l'Open Source ha, in generale, l'interesse a far sviluppare e progredire il software per offrire soluzioni più avanzate.

Chi vende hardware ha tutto l'interesse ad offrire il software a bassissimo costo per potersi concentrare sui propri prodotti; meno costa il software più hardware potrà vendere, questa è l'idea di fondo.
Questo vale sia per i grandi sistemi per i centri di calcolo che per il mondo embedded che sta vedendo Linux in forte crescita; la chiave per il successo di una piattaforma embedded dipende fortemente anche dalla diffusione e adottare Linux può essere la chiave per catalizzare l'attenzione verso una piattaforma cha ha immediatamente a disposizione migliaia di sviluppatori potenziali.

La comodità è una motivazione da non sottovalutare affatto; molte aziende non si interessano di informatica ne voglio farlo e non fa differenza che il software sia liberamente scaricabile e installabile, la soluzione chiavi in mano è l'unica accettata. Il cliente è disposto a pagare per non doversi preoccupare di nulla e avere la garanzia che tutto funzioni senza doversi impegnare sulla questione.
Questo può succedere sia perché l'informatica non è percepita come un valore, sia perché è stato deciso di esternalizzare completamente il settore, o talvolta anche semplicemente per pigrizia.

Chi detiene il copyright sul codice può decidere di rilasciare il software sotto due diverse licenze, una open e l'altra proprietaria, e decidere di differenziare le due versioni; il motivo per tenere questi due rami separati potrebbe essere la decisione di vendere a terzi il codice, per farlo integrare in altri programmi commerciali più ampi, o semplicemente per offrire due versioni con caratteristiche diverse.
Tipicamente la versione proprietaria offre feature indirizzate all'ambiente enterprise mentre quella open è più adatta per un uso in ambienti più ristretti.

Per potersi comunque avvantaggiare del modello open è necessario rilasciare la maggior parte del codice e tenere solo alcuni moduli proprietari da sviluppare completamente in-house.

Anche la gestione del marchio è una possibilità comune al software proprietario, altre aziende avranno infatti tutto l'interesse a farsi associare al marchio di una azienda prestigiosa, indipendentemente che offra software open o meno. I programmi di partnership sono una delle possibilità.

Ovviamente è possibile mescolare tutte queste possibilità per trovare la formula più adatta al prodotto.
Tipicamente vengono presentati quattro modelli, così come formalizzati dall'Open Source Initiative:
Support Sellers: chi appartiene a queste categoria offre gratuitamente il software per poter poi offrire tutti i servizi post vendita a pagamento, come assistenza, branding, integrazione ecc.
Loss Leader: il software open serve per preparare la strada ad altri prodotti proprietari dell'azienda
Widget Frosting: tipico esempio è la vendita di hardware, il software è open per consentire la più ampia diffusione possibile e abbassare i costi
Accessorizing: l'offerta non sul software ma su tutti i servizi accessori, dai manuali al merchandise.

Un altro aspetto che sfugge un po' alle logiche viste fino ad ora e che a volte viene del tutto ignorato è la diffusione di un prodotto come valore in se.
Il fatto che un prodotto sia molto diffuso e utilizzato è di per se un vantaggio sia pratico che politico :

- Poter vantare un certa base di installazioni garantisce una maggior credibilità davanti a potenziali investitori, il che può decidere il successo o meno di un progetto.
- Più la cerchia di utenti si allarga maggiore è il numero di ambienti e gli scenari in cui il software viene testato rendendolo di conseguenza più

solido; testare il software in ambienti simulati è un procedimento lungo e oneroso e a volte non particolarmente conclusivo, avere dei test reali è una grossa fonte di informazioni, oltre che un grosso risparmio di tempo e denaro.

- Come spesso è accaduto in altri progetti open, avere una larga base di utilizzo stimola l'adattamento e a volte addirittura il porting del software verso nuovi scenari di utilizzo inizialmente non previsti, allargando a sua volta il numero potenziali utenti.
- Potenzialmente significa anche avere più sviluppatori: oltre che per un semplice calcolo statistico, questo è dovuto all'aumento di interesse, anche da parte di terzi, verso il prodotto; più è alta la domanda maggiore sarà la probabilità che altri si interessino al progetto aprendo la strada a nuove collaborazioni e mercati.
- Più clienti utilizzano una soluzione più saranno i nuovi clienti che prenderanno in considerazione quella soluzione per i propri problemi; questo fenomeno può essere attribuito a numerosi fattori ma uno particolarmente interessante è il cosiddetto effetto a cascata[4] delle informazioni. Questo fenomeno sociale viene formalizzato una decina di anni fa dall'economista Daniel B. Klein in seguito allo studio di una serie di eventi avvenuti nel 1846. L'ingegnere americano George Geddes propose in quell'epoca le cosiddette strade di legno, ovvero delle strutture mobili utilizzate per compattare le strade americane fangose e difficilmente percorribile durante il periodo delle piogge; la soluzione, studiata per le condizioni di Salina, una piccola città nello stato di New York, ebbe un grande successo e l'idea prese piede. Nel giro di un paio di anni tutto lo stato di New York e poi tutto il Mid West aveva strade di legno. Il successo delle strade di legno non era dovuto all'effettiva validità della soluzione (chi ha mai visto oggi una strada di legno?) bensì all'effetto a valanga delle informazioni, appunto: quando è necessario adottare una soluzione a un problema complesso molto spesso si copia una soluzione già esistente di successo senza valutarla in effetti approfonditamente.

- Avere molti clienti vuol dire anche essere investiti di un'autorevolezza che può essere effettivamente spesa con i clienti e i partner e che può essere una fonte di guadagno. Questo aspetto in particolare ricade sotto quello che viene definito branding, ovvero la gestione del marchio: se si ha un marchio forte e riconosciuto si possono, ad esempio, attuare delle politiche di partnership a pagamento in cui sostanzialmente delle aziende pagano per vedere associato il proprio logo e nome a quello di una azienda famosa. Questa è una realtà consolidata e anche abbastanza abituale.

7 DataRiver

Nel 2008 è stato avviato il progetto per rendere MOMIS Open Source che ha portato alla nascita di DataRiver.

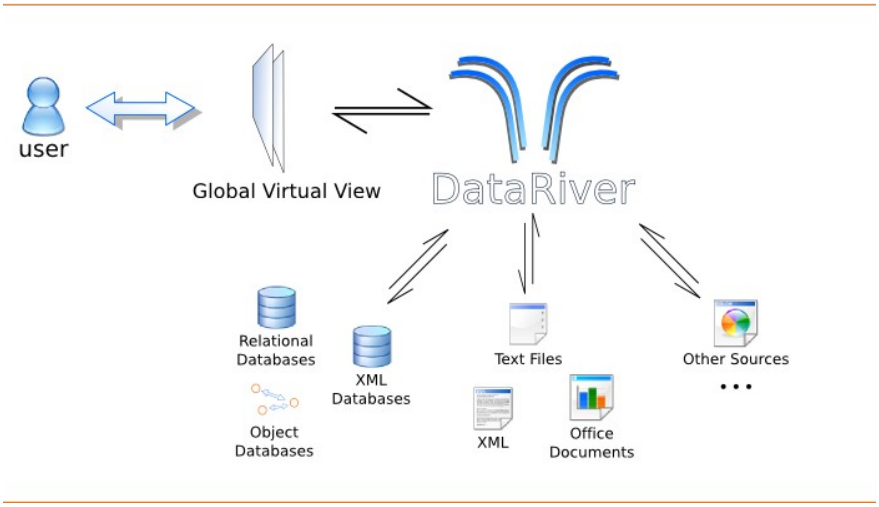


Figura 32: Schema di funzionamento di DataRiver

La commercializzazione di un software necessita di fare attenzione anche ad aspetti non propriamente tecnici ma tuttavia fondamentali; è indubbio infatti che il successo di un prodotto, al di la delle sue qualità intrinseche, è dato anche dall'immagine che se ne da, nonché dal modello commerciale adottato.

Lo sviluppo di questo progetto ha pertanto richiesto, oltre alle competenze tecniche proprie di un ingegnere informatico, anche di sviluppare tematiche più vicine al campo economico e del marketing.

Per un prodotto Open Source inoltre si aggiungono numerosi altri aspetti da tenere presenti perché, se per un software tradizionale lo scopo è di acquisire clienti, per un prodotto Open Source lo scopo è sia di acquisire clienti che sviluppatori.

Per facilitare l'accesso del progetto attraverso Internet la lingua di riferimento per il progetto è, come ci si poteva aspettare, l'inglese, pertanto è dato per scontato che qualsiasi documento sia pubblicato i questa lingua.

7.1 Nome e logo

Il nome e il logo sono il primo contatto dell'utente con il progetto e vi è stata per tanto dedicata particolare attenzione.

Per la creazione del nome sono seguiti tre approcci mentali differenti:
un'ipotesa vagliata è stata quella di un nome parlante , in cui il nome evocasse chiaramente la funzione del prodotto;
un'altra è stata quella invece di rifarsi all'italianità del prodotto, rievocando nel nome personaggi o simboli universali del nostro paese;
infine sono stati valutati anche nomi completamente distaccati da queste logiche e incentrati invece su una simbologia legata all'idea di forza e potenza.
Nel riquadro sono riportati alcuni dei nomi vagliati in questo processo di brain storming in cui è possibili identificare chiaramente questi tre approcci:

dataflow	datashed	babel
datariver	datalight	pito
datawork	carmine	connecto, connector
datawell	nyle	colligo
prodata	kem	explico
niagara	rosetta	cesare
datafluent	agregator	alessandro
dataflumen	flowjob	ottaviano
datafall	stige	augusto
easydata	lete	aurelio
datafinity	hydra	dragon
datarex	havel, dehavel, unhavel	typhoon

cyclone	dataway	rufus
datasquash	climby	cerberus
nebula	dataclimber	bellerofonte
phoenix	datagrit, hardgrit	chinesys
skyer	atlante, atlas	juggler, giaggler
lynx	calypso	datapillar
caracal	atlantys	concrete
cheetah	tytan	databurst
smilodon	brick, brik	

Dopo una prima scrematura sono stati estratti alcuni nomi che identificano meglio l'idea che si voleva esprimere e da questi sono stati elaborati alcuni loghi:

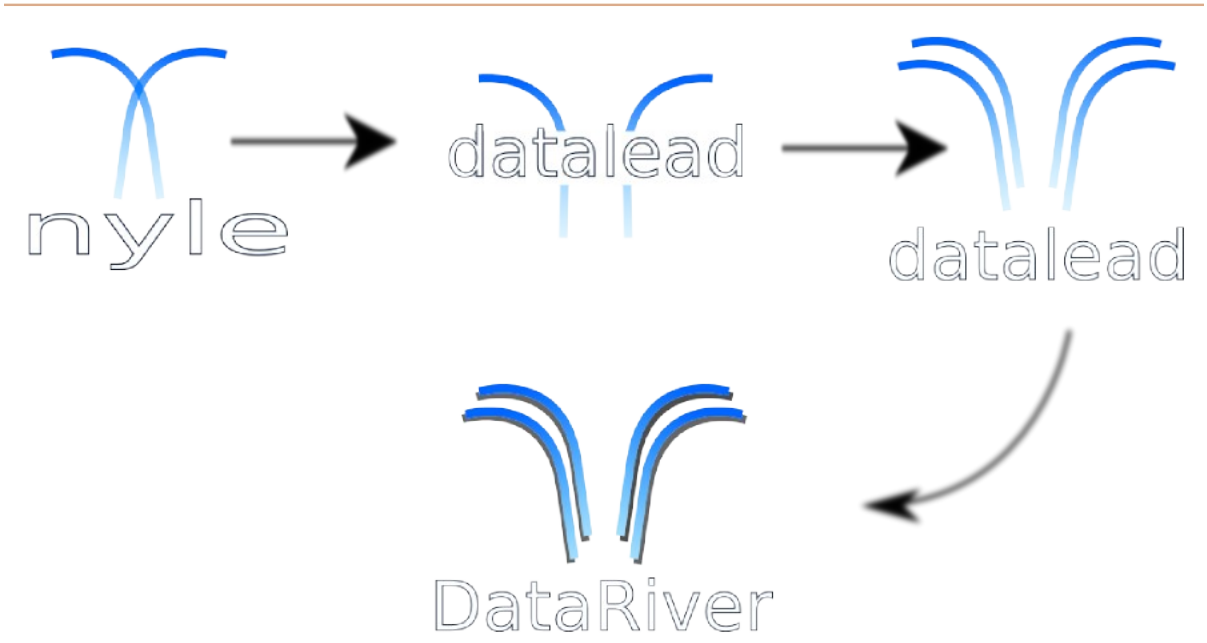


Figura 33: Evoluzione del logo per il progetto

Alla fine di questo processo creativo si è arrivati alla scelta del nome definitivo DataRiver.
Il logo è stato creato partendo dall'idea di unione e aggregazione, espressa

nel logo dalla convergenza delle linee sul nome; la palette di colori sull'azzurro risulta in tema col nome e offre una buona combinazione di colori per il web.

7.2 Documentazione

La documentazione di DataRiver è ereditata interamente da MOMIS che purtroppo è in difficoltà di quest'ambito; la documentazione esistente al momento è incompleta, per fare un esempio ci sono 2765 funzioni e più di 300 parametri non documentati, e parte in italiano e parte in inglese. Per consentire una formattazione automatizzata consistente da parte di Doxygen (si veda più avanti) sarà inoltre necessario ripulire i commenti dai tag HTML che sono stati talvolta inseriti.

Per un progetto Open Source la documentazione è di fondamentale importanza in quanto è la base di partenza di chiunque voglia partecipare allo sviluppo e sarà pertanto necessario molto lavoro per migliorare ed espandere questo aspetto.

Per DataRiver, come per qualsiasi progetto a dir la verità, sono state individuati 4 tipi di documentazione che si differenziano per oggetto, utente target e stile comunicativo:

- Documentazione Architetturale
- Documentazione Tecnica
- Documentazione End User
- Documentazione Pubblicitaria

La **documentazione architetturale** fa riferimento all'intera architettura del progetto, serve a dare uno sguardo dall'alto dei vari componenti senza entrare troppo nel dettaglio avvalendosi di immagini e grafici per spiegare le relazioni tra i componenti con un approccio scientifico indirizzato a un utente con competenze tecniche.

La **documentazione tecnica** al contrario è molto dettagliata e specifica, classico esempio sono i commenti all'interno del codice sorgente; l'utente di riferimento è indubbiamente un programmatore, lo stile è conciso e diretto.
La **documentazione così detta end user** è il classico manuale di utilizzo di un software indirizzato all'utente comune che non possiede conoscenze tecniche approfondite; lo stile è colloquiale e si utilizzano numerosi screenshot e aiuti visuali; oltre al classico manuale anche il forum e le FAQ possono essere un ottimo mezzo di diffusione della documentazione end user.
La **documentazione pubblicitaria** non è altro che pubblicità in tutte le sue forme: brochure, banner, volantini, filmati flash ecc.

Entrando nel dettaglio, andrà programmata la stesura di documentazione sia architetturale che pubblicitaria e sarà necessario un grosso sforzo per migliorare la documentazione tecnica. In particolare le attività su cui concentrare l'attenzione sono:

- Scrivere un documento sintetico di una o due pagine di introduzione all'architettura che spieghi semplicemente, anche attraverso immagini, gli step di esecuzione e il flusso dei dati; questo permette a chi è esterno al progetto di farsi velocemente e a grandi linee un'idea di come funzioni DataRiver. Per questo documento esiste già del materiale che va però scremato e ripresentato in una formula compatta.
- Scrivere le linee guida per la stesura della documentazione tecnica.
- Rendere fruibile la parte di documentazione tecnica che già esiste traducendo le parti in italiano e sistemando la parte in inglese, correggendo inoltre grammatica e sintassi.
- Completare la documentazione tecnica; la cosa più efficace sarebbe di assegnare la scrittura della documentazione a chi ha sviluppato il codice originariamente. In ogni caso essendo passato molto tempo tale attività sarà comunque difficoltosa e molto costosa in termini di

tempo.

- Stabilire le linee guida di accesso allo sviluppo del progetto; in pratica queste linee guida si manifestano per la maggior parte nelle regole di gestione del CVS: chi può postare? chiunque può postare per qualunque area del codice? un nuovo utente prima di poter postare deve dimostrare di aver capito il funzionamento del progetto, fornendo prima delle piccole patch?
- Scrivere un documento di introduzione al progetto in cui venga spiegato quali strumenti sono necessari per partecipare allo sviluppo, come ottenere il codice sorgente, come configurare CVS, le regole di testing e rilascio, la segnalazione di bug e così via.
- Consolidazione e miglioramento della guida alla compilazione di DataRiver.
- Stabilire la road map del progetto, specificando quali obbiettivi si vogliono raggiungere con le nuove release, i tempi e le metodologie di rilascio.
- Stesura di quella che si potrebbe definire dichiarazione d'intenti , ovvero una sorta di patto che il progetto stipula con la community garantendo un comportamento etico e di collaborazione reciproca.
- La documentazione pubblicitaria va completamente creata; per prima cosa sarebbero utili alcuni documenti in cui vengano presentati degli scenari di utilizzo in cui poter impiegare DataRiver ponendo ovviamente l'attenzione sulle problematiche che DataRiver è in grado di risolvere e non su aspetti tecnici.
- Prima di poter essere rilasciato DataRiver ha bisogno di alcune modifiche all'interfaccia grafica, sarà pertanto necessario adeguare la documentazione end-user.
- Benché non sia propriamente documentazione il forum è comunque un centro informale di aggregazione di informazioni, nonché il primo punto di contatto da chi si avvicina al progetto; è importante quindi definire con chiarezza le linee guida di gestione del forum per poter garantire tempi di risposta accettabili.

7.3 Licenza

All'interno di una strategia di commercializzazione la licenza occupa una posizione determinante ed è pertanto da valutare attentamente.
Attualmente MOMIS utilizza una licenza che ne permette il libero utilizzo, limitatamente ad un ambito accademico e di ricerca, ma non consente l'accesso al codice sorgente; per dare una svolta open al progetto sono state vagliate tutte le maggiori licenze Open Source concentrando in fine l'attenzione sulla GPL versione 2.
All'interno delle licenze Open Source, discusse nel capitolo precedente, la più efficace per incentivare lo sviluppo condiviso e proteggersi dal Free Riding è proprio la GPL che consente di avere accesso al codice sorgente, modificarlo e ridistribuirlo con l'obbligo di mantenere però la stessa licenza.
Questa licenza consente di abbattere qualsiasi barriera e difficoltà di collaborazione nel mondo accademico ma contemporaneamente assicura che nessuna azienda possa avvantaggiarsi del lavoro svolto senza dare nulla in cambio.
Un'ulteriore scelta da effettuare è quale versione della GPL adottare: la versione 2 è attualmente la più diffusa, con migliaia di progetti attivi che la utilizzano, e questo mette a disposizione potenzialmente un enorme volume di codice già pronto che può essere integrato liberamente. Sarebbe pertanto alquanto invalidante escludersi un bacino così ampio di applicazioni adottando la nuova GPL versione 3, anche se essa sembra essere la scelta più ovvia in futuro.
WordNet, come accennato precedentemente, è un tool esterno, sviluppato dall'Università di Princeton, ma non apporta nessun nuovo problema alla scelta della licenza in quando rilasciato con una licenza simil BSD (vedi capitolo successivo); la licenza adottata in pratica impone unicamente di mantenere la nota di copyright e citare gli autori, così come recitato nel seguente estratto:

Permission to use, copy, modify and distribute this software and database and its documentation for any purpose and without fee or royalty is hereby granted, provided that you agree to comply with the following copyright notice and statements, including the disclaimer, and that the same appear on ALL copies of the software, database and documentation, including modifications that you make for internal use or for distribution.
La breve licenza di WordNet può essere visionata per intero in appendice B.

7.4 Strumenti per la community

La fase fondamentale che determina il successo o il fallimento di un progetto Open Source è la costruzione della propria community; una community attiva e numerosa, come visto anche nei capitoli precedenti, è un vantaggio competitivo di enorme valore e contribuisce inoltre in maniera determinante alla costruzione dell'immagine del prodotto e dell'azienda che promuove il progetto.
La gestione di una community vuol dire innanzitutto essere fornitori di comodità, diventa quindi immediata la necessità di dotarsi di strumenti organizzativi e di gestione per permettere ai membri della comunità di concentrarsi sul progetto e abbattere le difficoltà di comunicazione.
Per coadiuvare le attività di una eventuale community sono stati identificati alcuni strumenti indispensabili che sono stati successivamente implementati.
Le problematiche da risolvere, abbastanza comuni in un progetto open, sono la promozione del progetto, la comunicazione tra i membri, attraverso canali formali e informali, la gestione della documentazione, la gestione del codice sorgente e il tracking dei bug.
Internet è senz'altro lo strumento principe per questo tipo di progetti ed era quindi scontato che tutti gli strumenti considerati dovessero essere orientati al web ed utilizzabili direttamente all'interno di un browser.
Il web è stato scelto come canale di comunicazione e promozione e quindi si è provveduto, dopo una analisi sintetica dei Content Management System

Open Source in circolazione, a creare il portale del progetto, cercando di porre l'accento anche sull'aspetto estetico.
Di seguito viene riportato anche il frutto delle analisi dei prodotti open per la gestione della documentazione e di gestione / condivisione del codice e del bug tracking.
Gli strumenti scelti alla fine sono stati Drupal con l'aggiunta di un certo numero di moduli, Doxygen, CVS e BugZilla.

7.4.1 Drupal

Un Content Management System, CMS d'ora in avanti, è un sistema per la gestione dei contenuti e della struttura di un sito web in maniera automatizzata; ogni CMS ha lo scopo di facilitare il lavoro dei web master diminuendo la complessità della gestione di un sito con conseguente risparmio di tempo.
In generale i CMS offrono degli strumenti di supporto alla creazione della struttura del sito e dei contenuti (Content Management Application), nonché alla presentazione degli stessi (Content Delivery Application).
I CMS separano le informazioni dalla loro presentazione e le trattano in maniera separata; ciò permette una personalizzazione facile e immediata dell'aspetto grafico del sito e delle modalità di fruizione dei contenuti.
Esistono numerosi CMS e molti di questi sono liberi, il primo passo dunque è quello di scremare tra tutte le possibilità offerte; è possibile ottenere una lista completa dei CMS open source continuamente aggiornata su www.opensourcecms.com.
Le caratteristiche di valutazione dei CMS sono state innanzitutto la qualità del progetto in se: maturità del progetto, diffusione, facilità di utilizzo, linguaggio di sviluppo, qualità della documentazione, quantità di moduli di terze parti e sicurezza; è stata considerata la presenza di un forum e di una chat è una caratteristica imprescindibile.
Il linguaggio di riferimento scelto è il PHP, sia perché è molto diffuso e

conosciuto nonché facile ed immediato, quanto perché permette una integrazione con Apache banale.
Dopo una breve cernita l'attenzione si è focalizzata sui CMS più diffusi: PHP-Nuke, Joomla e Drupal.
PHP-Nuke ha goduto in passato di una enorme successo, nonché di una certa fama, sia per la maturità del progetto sia per la diffusione dello stesso e per la moltitudine di cloni e fork (CPG-Nuke, Post-Nuke, MyPhp- Nuke, Xoops, solo per citarne alcuni).
Attualmente PHP-Nuke è stato sorpassato in popolarità e funzionalità sia da Joomla che da Drupal, che risultano essere i due protagonisti del mondo dei CMS Open Source.
Joomla è un ottimo prodotto sotto molti aspetti, buona velocità e performance, pulizia del codice, politica di sviluppo, licenza e attività della comunità, ma purtroppo soffre di alcune manchevolezze; sono infatti tutte gli accessori che stanno intorno a questo prodotto a penalizzarlo maggiormente.
In particolare la documentazione risulta di qualità non adeguata al tipo di progetto, rendendo tutto inutilmente più difficoltoso.

Figura 34: Homepage del sito di DataRiver

Drupal è un ottimo CMS che offre tutte le funzionalità richieste e ha inoltre una community folta e attiva; la qualità della documentazione è molto buona, e sono presenti online numerose guide per usare, personalizzare e sviluppare Drupal che trattano argomenti sia da newbie che da pro.
I numerosissimi moduli aggiuntivi mettono poi a disposizione tutte le funzionalità che venivano richieste tra cui l'esportazione delle pagine del sito in PDF e il forum.

Nell'immagine è possibile vedere l'aspetto della homepage del sito creato per DataRiver; si è cercato di sviluppare l'aspetto grafico del sito facendo sempre riferimento al nome e utilizzando la palette di colori del logo per dare uniformità e continuità.

Drupal, come detto, consente di creare direttamente online le pagine web del sito senza bisogno di scrivere codice e, attraverso il modulo TinyMCE in combinazione con il modulo IMCE, si ha a disposizione un editor completo con supporto alle immagini direttamente online con anche il correttore ortografico.
La gerarchia degli utenti integrata in Drupal consente di gestire la creazione, la revisione e la pubblicazione dei contenuti in maniera granulare, permettendo uno sviluppo collaborativo e condiviso del sito; il version control integrato sui documenti poi permette di tenere traccia delle modifiche apportate ad un documento così da essere sempre in grado di saper chi ha lavorato su quale documento.
Attraverso il modulo Pdfview è inoltre possibile aggiungere l'esportazione delle pagine in PDF. L'esportazione è fatta a runtime, per così dire, quindi quando si modifica una pagina web la versione PDF è automaticamente aggiornata.

Attraverso il modulo GraphStat è possibile ottenere grafici statistici sull'andamento del sito: numero di utenti, visite, ma anche attività degli utenti o parti del sito particolarmente attive.

Come tutti gli strumenti scelti anche Drupal è Open Source.

Per tutti i tipi di documentazione al di fuori di quella tecnica Drupal è risultata la scelta più efficace in quanto consente di gestire tutto online, in maniera condivisa, e dando la possibilità di delegare le funzioni di creazione

e gestione dei contenuti.

7.4.2 Doxygen

Per poter usare con profitto la documentazione tecnica è imprescindibile dotarsi di strumenti per automatizzarne l'estrazione dal codice sorgente, la gestione e la formattazione.
Gli strumenti presi in esame sono stati una decina ma per non dilungare ulteriormente la fase di analisi è riassunta nelle poche righe qui di seguito:

- AutoDoc: soluzione quick and dirty, facile da configurare ma poco flessibile e poco utilizzato.
- Autodoc doclet: doclet sviluppata ad hoc per MOMIS e utilizzata in passato; già funzionante ma poco flessibile e espandibile.
- POD: plain old text, metodo usato per il Perl; permette l'esportazione semi automatica in HTML, PDF, TEX e man pages, facile da utilizzare, possibilità di scrivere commenti in html. Se non si scrivono i commenti direttamente in HTML non c'è nessuna formattazione avanzata, niente tabelle, immagini o gestione dei font.
- TwinText: proprietario.
- RoboDoc: evoluzione di Autodoc a cui aggiunge il supporto all'XML e la possibilità di esportazre in PDF e HTML; presenta tutti i problemi di Autodoc.
- HeaderDoc: sviluppato dalla Apple è molto simile concettualmente a RoboDoc e AutoDoc, disponibile solo per Mac e Linux.
- Kelp: molto semplice da utilizzare ma i commenti vanno tenuti in documenti separati dal codice.
- JavaDoc: standard de facto per la documentazione tecnica limitatamente al codice sorgente scritto in Java; esportazione in tutti i formati più conosciuti.
- Doxygen: multi-piattaforma e molto utilizzato nelle comunità open tra

cui quella KDE e di Drupal; supporto a diversi linguaggi di programmazione tra cui Java, C e Python, facile da utilizzare, esporta automaticamente in tutti i formati più conosciuti come: HTML, PDF, XML, CHM, TEX, RTF e altri. Integra un motore di ricerca per la documentazione scritto in PHP ed è possibile personalizzare tramite CSS l'aspetto dell'HTML generato. Permette la generazione automatica di diagrammi e schemi estratti dalla relazioni tra le classi del codice sorgente.

Nella figura sottostante vediamo lo schema di funzionamento di Doxygen: al centro c'è ovviamente Doxygen stesso che legge il codice sorgente e lo parsa; volendo è possibile passare dei footer, header o immagini personalizzate che verrano integrate all'interno della documentazione generata automaticamente dal sistema.
Attraverso il file di configurazione si specifica il tipo di documentazione che verrà generata, HTML piuttosto che XML o PDF, l'aspetto grafico e tutti gli altri dettagli.
Attraverso i Doxytag è possibile estendere la documentazione integrandoci file esterni al codice.

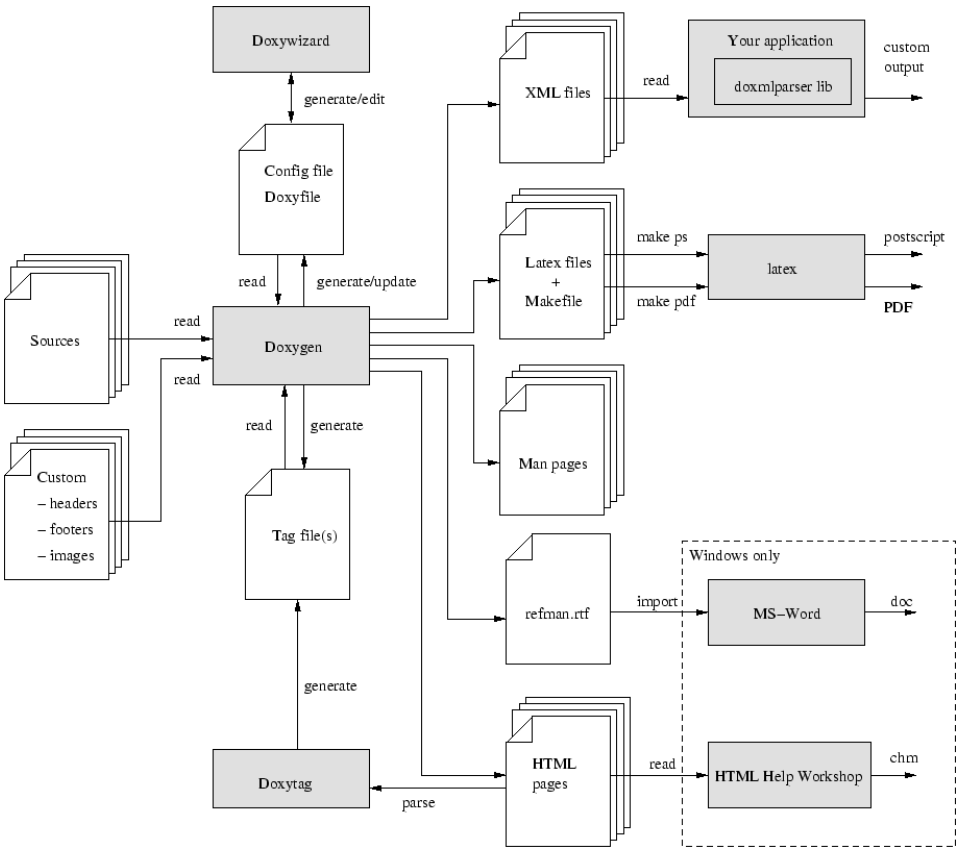


Figura 35: Flusso delle informazioni in Doxygen

Qui di seguito è riportato un estratto del file di configurazione scritto per la documentazione di DataRiver con le direttive di maggiore interesse:

```
#---------------------------------------------------------------------
--
# Project related configuration options
#---------------------------------------------------------------------
--

DOXYFILE_ENCODING  = UTF-8
PROJECT_NAME           =  DATARIVER
PROJECT_NUMBER         = 0.1
OUTPUT_DIRECTORY       = /var/www/datariver
CREATE_SUBDIRS         = NO
OUTPUT_LANGUAGE        = English
BRIEF_MEMBER_DESC      = YES
INPUT                  =  /home/fry/workspace/momis/momis/src/
FILE_PATTERNS          = *.java
RECURSIVE              = YES
EXCLUDE_SYMLINKS       = NO
GENERATE_HTML          = YES
HTML_OUTPUT            = html
HTML_FILE_EXTENSION    = .html
GENERATE_LATEX         = NO
GENERATE_RTF           = NO
GENERATE_MAN           = NO
GENERATE_XML           = NO
GENERATE_PERLMOD       = NO
ENABLE_PREPROCESSING   = YES
CLASS_DIAGRAMS         = YES
HAVE_DOT               = YES
CLASS_GRAPH            = YES
COLLABORATION_GRAPH    = YES
UML_LOOK               = YES
```

Il file di configurazione qui presentato è incompleto, l'originale contiene più di duecento direttive ed è quindi evidente che può essere alquanto noioso scrivere e tenere traccia di tutte.

Un primo trucco per aggirare il problema è quello di creare un file di configurazione standard e andare quindi a modificare solo le direttive di interesse, che normalmente non sono più di una ventina.

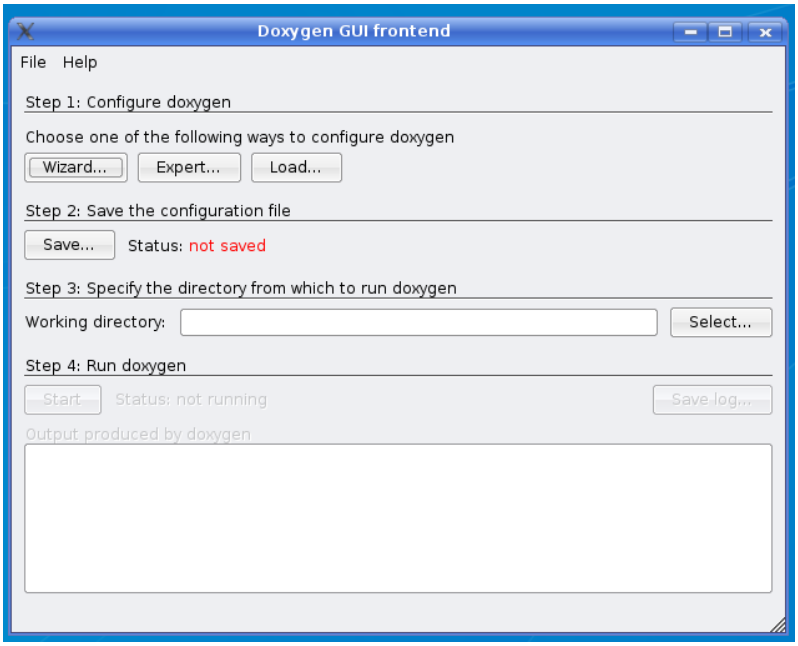


Figura 36: DoxyWizard

Una volta installato sulla macchina Doxygen è sufficiente dare il comando doxygen -g per generare un file di configurazione standard chiamato con fantasia Doxygen .Un'alternativa è quella di utilizzare DoxyWizard, mostrato nella figura qui sopra, un programma grafico che guida l'utente attraverso la fase di creazione del file tramite un wizard passo passo.

7.4.3 CVS e BugZilla

Per la gestione delle revisioni e il tracking dei bug si è optato per CVS e BugZilla.

CVS è lo strumento storico in questo campo ed è stato scelto rispetto al più recente Subversion perché già conosciuto e utilizzato dal gruppo di sviluppo originale di MOMIS e perché si integra meglio all'interno di Drupal grazie al

modulo CVS integration.
Anche BugZilla è considerato uno standard nel mondo open ed è stato scelto perché ampiamente diffuso.
In futuro si effettuerà la sperimentazione di Mantis, in forte crescita e destinato a soppiantare BugZilla.

7.5 Evoluzione di MOMIS verso DataRiver

Prima di poter essere reso fruibile DataRiver necessita di alcuni ritocchi che lo traghettino dal mondo accademico al mondo commerciale, migliorandone l'usabilità e la praticità di utilizzo. I miglioramenti da apportare al progetto non sono esclusivamente di natura tecnica ma anche organizzativa e comunicativa. Le aree su cui intervenire sono:

7.5.1 Installer

Sarebbe necessario creare un metodo standardizzato e coerente per distribuire il software in maniera pacchettizzata. Trattandosi di un software Java non è strettamente necessario un installer ma è comunque utile sviluppare un metodo di distribuzione click and run dell'applicativo, così da poter essere installato e utilizzato, almeno come test, facilmente.
Alcuni aggiustamenti vanno in ogni caso apportati agli script e al sistema di avvio in quanto non funzionano correttamente sotto Linux, è facile trovare path non corretti, ad esempio, che richiedono l'intervento manuale dell'utente.
Allo stesso modo è da sistemare la gerarchia delle cartelle per razionalizzare la disposizione dei file e degli script e adeguarla alle consuetudini del mondo Unix: ad esempio gli script per lanciare il Designer si trovano attualmente nella cartella var, mentre in quella bin sono conservati gli script utili ad Ant

per la compilazione dei sorgenti.

7.5.2 Interfaccia grafica

L'interfaccia necessita di aggiustamenti sia per quello che riguarda l'usabilità che il banale aspetto estetico. E' necessario ripensare il sistema di etichettatura dei pulsanti, sostituendo ad esempio riferimenti ad aspetti tecnici dell'architettura di DataRiver con riferimenti alle funzionalità del bottone o chiarificandone l'effettiva azione associata.

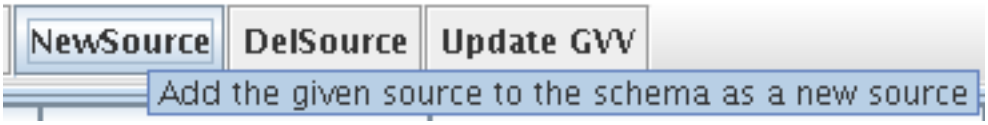


Figura 37: Il bottone riporta l'etichetta NewSource ma come azione Add the given source

Come si vede in figura il bottone NewSource in realtà aggiunge la sorgente appena creata allo schema, sarebbe più efficace sostituire l'etichetta con AddSource.

Vanno ripensati anche le finestre e i messaggi di errore spesso criptici e non realmente informativi; come si vede in figura il messaggio di errore non aiuta minimamente l'utente a risolvere il problema.

Figura 38: Messaggio di errore di scarso aiuto

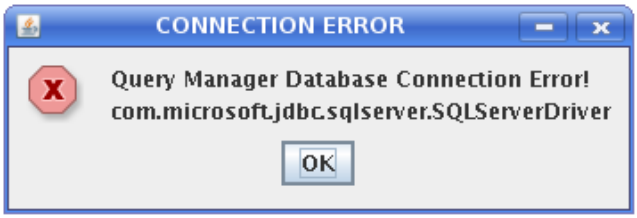


Figura 39: Errore di connessione

Figura 40: Errore generico

Ci sono inoltre alcune sviste macroscopiche, come la stringa di connessione al database che viene visualizzata in chiaro mostrando la password.

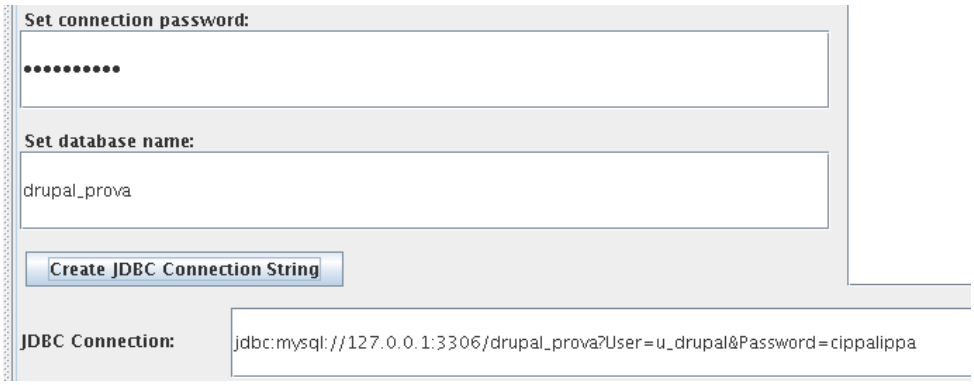


Figura 41: La password viene offuscata ma compare in chiaro nella stringa di connessione

Una cosa simile ma più grave avviene anche per i log: le exception espongono nei log tutta la stringa di connessione mostrando anche in questo caso la password:

```
wrapper class name:
it.unimo.dbgroup.momis.communication.core.jdbc.WrapperJdbcCore_MySql
 - Getting connection with [jdbc:mysql://127.0.0.1:3306/drupal_prova?
User=u_drupal&Password=cippalippa]...
[2008.04.01 12:19:51] Getting source description...
```

Da migliorare è la configurazione dei cluster: chi si trova a dover configurare i cluster, in ultima analisi, procede a tentativi inserendo valori casuali e cercando il risultato migliore impiegando molto tempo. Una procedura semi-guidata che indirizzi verso la scelta più opportuna dei valori, magari mostrando in contemporanea diversi risultati con vari valori per la creazione dei cluster, può essere la soluzione.

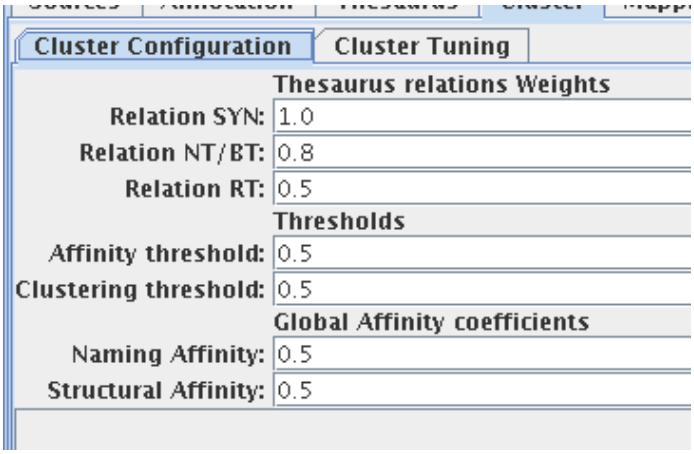


Figura 42: Configurazione dei cluster

7.5.3 Performance

Il query manager in particolare soffre di una lentezza eccessiva che, in caso la mole dei dati gestiti sia importante, lo può rendere inutilizzabile nel concreto. Se è normale, in linea di principio, un tempo di esecuzione delle query di qualche decina di minuti per i sistemi ETL, in cui le query sono generalmente complesse e rivolte alla business intelligence, non lo è se

l'obbiettivo che ci si prefigge è di diventare il punto di accesso comune ai dati. Questo pone anche una nuova problematica: un utente malevolo potrebbe infatti, con l'esecuzione di numerose query complesse in contemporanea, portare al denial of service più sorgenti dati e DataRiver stesso.
Potrebbe essere interessante introdurre un sistema di reporting e statistiche per tenere traccia delle prestazioni del sistema così da poter avere una base su cui operare eventuali aggiustamenti e potenziamenti; eventualmente si potrebbe offrire un software esterno a pagamento per il tuning del sistema.
In via temporanea si potrebbe implementare un sistema di limitazione delle risorse per proteggere il sistema da eventuali attacchi.

7.5.4 Supporto multilingua

Essendo un prodotto destinato a clienti con esigenze complesse e di alto profilo il supporto multilingua risulta essere un requisito abbastanza comune. Benché possa sembrare banale questa è una criticità di difficile soluzione; se infatti aggiungere il supporto multilingua all'interfaccia grafica non è immediato, ma comunque possibile, il supporto multilingua nell'architettura è tutto da valutare. Attualmente WordNet fornisce il supporto esclusivamente alla lingua inglese, il che è limitante di per se, ed inoltre pone alcuni risvolti inaspettati: accettando l'ipotesi che le sorgenti dati siano state etichettate in inglese, all'operatore è richiesto in ogni caso non solo di sapere usare il software ma anche di avere una buona conoscenza della lingua inglese, per non incappare in errori di interpretazione causati dell'ambiguità intrinseca del linguaggio.
Una possibile soluzione potrebbe essere l'integrazione di MultiWordNet che offre il supporto anche ad altre lingue, oltre all'inglese. La limitazione maggiore di questo sistema risulta però il numero di parole disponibili in italiano, circa un quarto di quelle disponibili in inglese.
E' stata già avviata ad ogni modo dal DBGroup una collaborazione con Expert System per ovviare a questo problema.

7.5.5 Refactoring

Il codice è stato sviluppato in un lasso di tempo molto ampio da diverse persone e richiede pertanto un buona dose di refactoring per razionalizzare il codice e rendere più facile lo sviluppo da parte di persone esterne al progetto.
All'interno delle operazioni di refactoring uno sforzo importante dovrebbe essere protratto per commentare il codice, aspetto fondamentale per facilitare l'accesso di nuove persone.

7.5.6 Aggiornamento

Al momento non c'è alcuno strumento che permetta di aggiornare il software in maniera autonoma; sarebbe utile progettare e implementare un sistema automatico di update almeno per le patch minori. Trattandosi di un software complesso e critico è accettabile pensare che per gli aggiornamenti più corposi sia necessaria una procedura manuale.

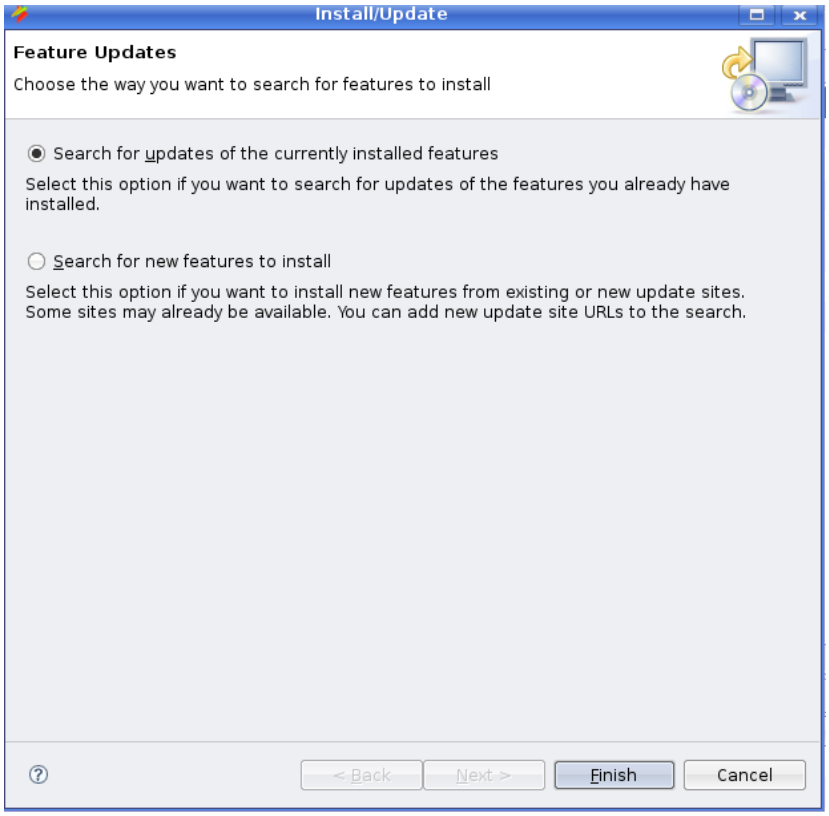


Figura 43: Scelta del tipo di aggiornamento da effettuare

Un sistema che si potrebbe prendere ad esempio è quello di Eclipse che offre sia la possibilità di fare un aggiornamento di mantenimento sia di aggiunta di nuove feature:

Scelto il tipo di aggiornamento una semplice finestra mostra le operazioni in corso:

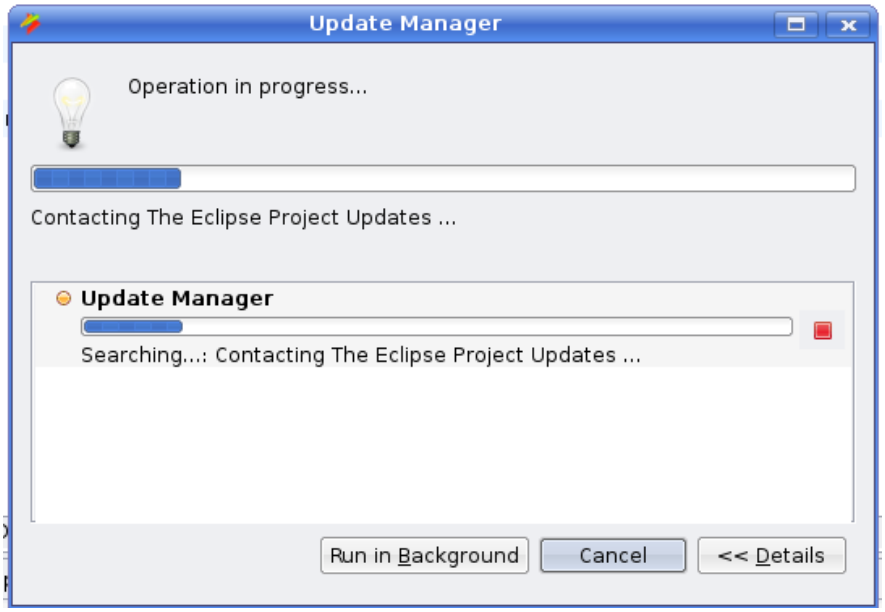


Figura 44: Update Manager di Eclipse

7.5.7 Riattivare lo sviluppo

Lo sviluppo di DataRiver procede a rilento, nell'ultimo anno i post nel CVS del progetto sono stati circa una decina di cui la maggior parte si tratta di patch di piccola entità. L'Open Source è senz'altro una risorsa sotto questo aspetto e un grosso aiuto nell'innescare l'innovazione, ma non bisogna incappare nell'errore di pensare di poter delegare alla comunità lo sviluppo in toto. In particolar modo nelle fasi iniziali di un progetto è richiesto un grosso sforzo, maggiore di quello che sarebbe richiesto dall'avviare un progetto proprietario, perché, come già ripetuto più volte, si cerca di avvicinare al progetto sia nuovi utenti che nuovi sviluppatori.

7.5.8 Comunicazione

MOMIS è stato un progetto puramente accademico ed è quindi ora richiesto di rielaborare tutta quella sovrastruttura informativa che gravita intorno al

progetto, passando da un ottica tecnologica e di ricerca ad un ottica commerciale incentrata sul prodotto. Sono da studiare e realizzare documentazione commerciale, brochure e la promozione in generale.

7.5.9 Test e Debugging

Benché il codice esista già da parecchio tempo, come precauzione, prima di rendere pubblico DataRiver è fondamentale eseguire un ciclo di test e debugging approfondito per non dare una cattiva immagine del progetto fin dall'inizio.
Avendo al momento delle carenze per quel che riguarda l'assistenza è molto importante dedicare maggiore attenzione a questa fase per poter ridurre al minimo i malfunzionamenti e quindi le richieste di intervento da parte del cliente.

7.5.10 Sicurezza

Non esiste un sistema di gestione degli accessi ne di provisioning / deprovisioning centralizzati, una volta che si è in grado di operare su DataRiver non ci sono ulteriori discriminazioni e si può intervenire su tutte le informazioni che questo gestisce; questo può essere una grande limitazione per ambienti complessi e articolati come quelli enterprise. E' senz'altro interessante pianificare l'introduzione di qualche sistema di autenticazione single sign-on, autorizzazione e visibilità dei dati. Avendo accesso a molte informazioni aziendali DataRiver è destinato a diventare molto probabilmente oggetto di attacchi informatici ed è pertanto richiesto di studiare attentamente la sicurezza del programma, facendo attenzione in particolare agli attacchi interni che sono quelli più pericolosi e inoltre più facili da portare a termine.
Gestendo una grande quantità di dati ed essendo possibile effettuare query molto complesse è abbastanza facile al momento consumare tutte le risorse

e innescare un denial of service dell'intero server su cui gira DataRiver, volontariamente o anche involontariamente; in particolare il sistema di gestione dei risultati parziali tramite tabelle temporanea può saturare facilmente la memoria.
Sviste inoltre come riportare le password in chiaro nell'interfaccia o nei log (si veda prima in questo paragrafo) possono distruggere l'immagine del prodotto e vanificare tutti gli sforzi protratti nello sviluppo.
Un aspetto delicato e da valutare è in particolare la visibilità dei dati soprattutto se non vi è alcuna distinzione nelle sorgenti stesse; è possibile pensare che chi utilizza direttamente le sorgenti abbia diversi privilegi di chi vede i dati integrati da DataRiver ed è possibile quindi che ci siano dati sensibili all'interno della sorgenti che non vadano esposti. In questo caso un livello di dettaglio che si limiti a consentire o negare l'accesso alla sorgente nella sua interezza può non essere sufficiente.
Nell'immagine vediamo una schematizzazione di massima del

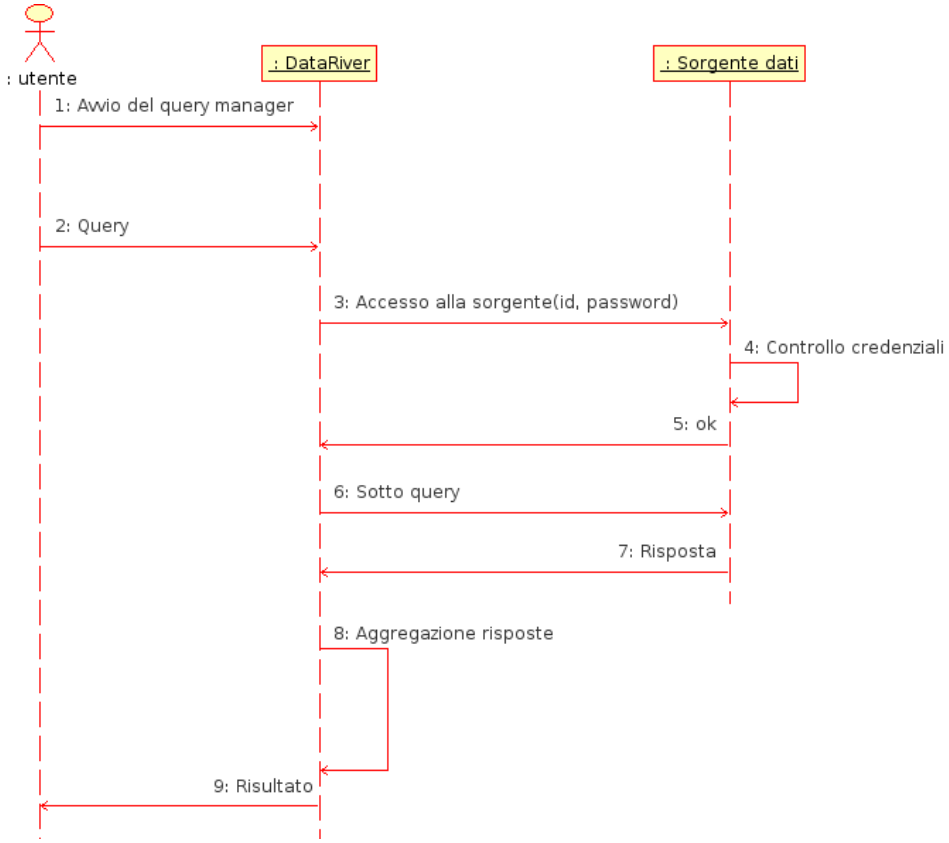


Figura 45: Schema di massima dell'esecuzione di una interrogazione

funzionamento attuale; una volta che qualcuno ha impostato le connessioni alle sorgenti dati, fornendo ad esempio l'Id e la password di connessione al database, chiunque abbia accesso a DataRiver ha automaticamente accesso a tutti i dati; è DataRiver infatti che mantiene in memoria le credenziali e che dopo si occupa di rinviarle alle sorgenti senza alcun controllo sull'utente.

Come primo passo sarebbe opportuno introdurre un sistema di autenticazione degli utenti e in base a questo filtrare i dati che possono essere effettivamente mostrati:

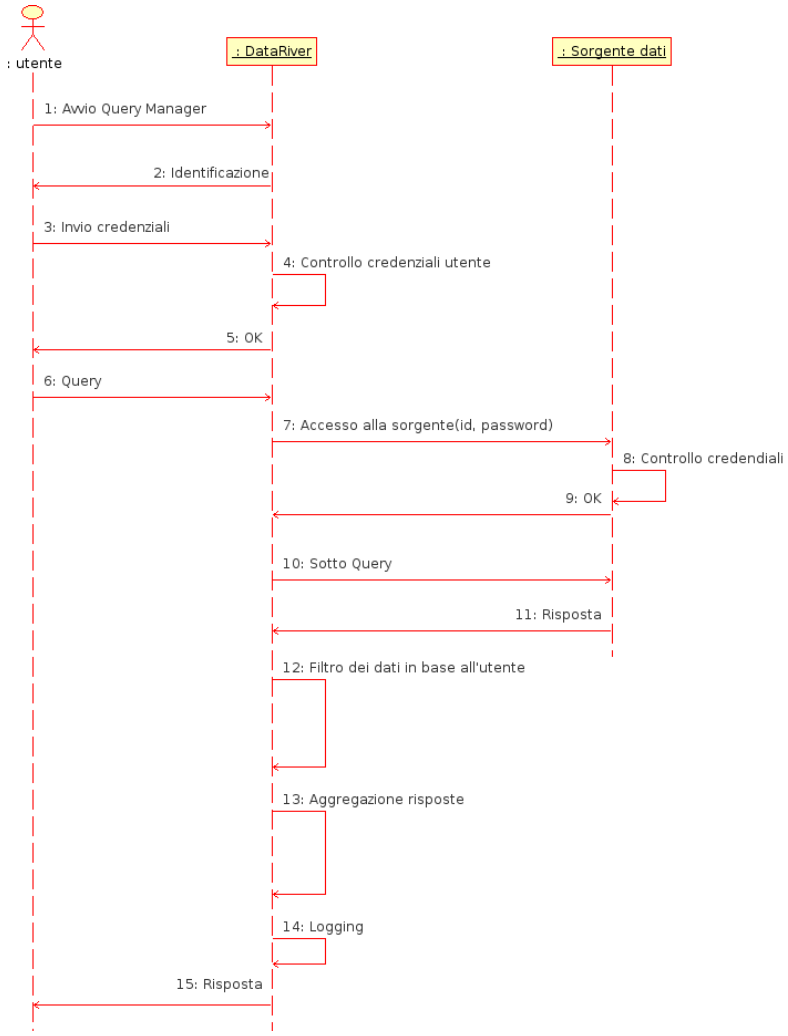


Figura 46: Autenticazione e Filtro dei dati in base all'utente

7.5.11 Gestione delle sorgenti

Nel momento in cui si gestiscono più sorgenti dati nasce il problema di come affrontare l'indisponibilità di una sorgente; quando un database non è raggiungibile, perché in manutenzione, per un problema di rete o qualsiasi sia il motivo, è necessario scegliere se non dare alcuna risposta, dare una risposta parziale (sempre che sia possibile) segnalando eventualmente la cosa all'utente o, scelta che si rivelerà prima o poi catastrofica, semplicemente ignorare il problema.
C'è inoltre da fare un ulteriore distinguo: un conto infatti è avere una sorgente non raggiungibile fin dall'inizio, un conto è avere una sorgente che crolla a metà dell'esecuzione; se nel secondo caso non c'è molto da fare, nel primo invece si potrebbe implementare un semplice sistema di controllo delle sorgenti che comunichi in anticipo all'utente quali sorgenti siano disponibili e quali no.
Al momento non esiste alcun tipo di controllo, semplicemente l'utente arriva al lancio della query per poi ricevere un errore, quando invece si sarebbe potuto avvisare subito l'utente consentendogli di risparmia tempo.
Se si decide di dare una risposta all'utente anche nel caso in cui una o più sorgenti fossero indisponibili va valutato anche come segnalare questo problema, in maniera tale che l'utente sia sempre consapevole che la risposta data non è completa.
La possibilità di malfunzionamento di almeno una delle sorgenti aumenta esponenzialmente con il numero delle sorgenti stesse, l'affidabilità del sistema nella sua interezza è quindi non del tutto sotto il controllo di DataRiver; potrebbe essere utile implementare un sistema di statistiche e reporting che mantenga lo storico della situazione delle sorgenti, così da poter verificare quale effettivamente sia stata l'efficenza del sistema globalmente e quali siano i punti più fragili.

7.5.12 Complessità

Il query manager offre la possibilità di interrogare le varie sorgenti dati ad un livello di dettaglio vicino a quello che si otterrebbe con un database attraverso l'SQL; questa feature, molto potente, è una caratteristica distintiva rispetto a tutti gli altri prodotti testati ma può risultare troppo complessa per gli utenti finali. Se infatti è più che lecito aspettarsi che l'integration designer sia esperto delle sorgenti dati e di DataRiver, l'utente finale, che vuole solo accedere ai dati, può non avere assolutamente idea del funzionamento del sistema o di come incrociare i dati. Potrebbe essere interessante proporre un metodo guidato per realizzare delle interfacce non modificabili da esporre direttamente agli utenti con la query già preconfezionata. Questo potrebbe vanificare in parte lo sforzo attuato per realizzare un query manager così avanzato, ma il rischio concreto è che, una volta implementato, il sistema non venga utilizzato perché troppo complesso.
Per facilitare ulteriormente l'utilizzo e proteggere gli utenti da errori involontari sarebbe utile separare effettivamente il Query Manager e distribuirlo come un programma separato, in maniera tale che anche volendo chi esegue una query non possa agire sulla GVV.

7.5.13 Esportazione dei dati

Sarebbe molto utile implementare un sistema di esportazione dei dati al di fuori di DataRiver in un formato, standard come ad esempio XML o, a malincuore, in un formato non standard ma utilizzato come Excel. Non poter esportare facilmente i dati può essere considerata una grossa limitazione, in particolare se si decide di intraprendere una commercializzazione rivolta alla business intelligence (si veda prossimo paragrafo).
Questo aspetto potrebbe essere fortemente sviluppato e si potrebbero persino offrire soluzioni avanzate a pagamento che facilitino il reporting.

7.5.14 Indipendenza da SQL-Server

Per porre l'accento sulla natura Open Source e multi-piattaforma del progetto, nonché per non doversi sobbarcare ogni volta del costo delle licenze, sarebbe vantaggioso slegarsi da SQL-Server e optare per un DBMS di appoggio Open Source.
La dipendenza da SQL-Server rende di fatto obbligatorio avere a disposizione una costosa infrastruttura Microsoft anche solo per testare DataRiver; nel caso di installazioni semplici mono-macchina questo impone di fatto l'utilizzo di DataRiver esclusivamente su Windows.
La scelta quindi ricade su due possibilità: MySQL e PostgreSQL.
MySQL non supporta nativamente il full outer join, anche se è possibile implementarlo con alcuni work around ben noti attraverso l'utilizzo di union, con una notevole diminuzione delle performance, e la scelta ricade pertanto su PostgreSQL.
La recente acquisizione di MySQL da parte di SUN inoltre pone ancora alcuni dubbi sulla direzione che prenderà lo sviluppo di questo DMBS mentre PostgreSQL si prepara inoltre a ricevere una iniezione di liquidità da parte di IBM che ha annunciato un investimento di dieci milioni di dollari; ciò dovrebbe garantire un ulteriore miglioramento del progetto in tempi brevi.
Il porting a PostgreSQL non dovrebbe risultare particolarmente difficoltoso visto che offre nativamente completo supporto al full outer join e alle altre feature che offre SQL-Server.

7.5.15 Supporto

Da pianificare per intero è la gestione del supporto al cliente: rimanendo sempre nell'ottica di puntare a clienti enterprise, che verosimilmente sono quelli che hanno maggiore bisogno di fare data integration, il supporto è il primo requisito per poter proporre qualsiasi soluzione.
Per cercare di ridimensionare il problema del supporto è imperativo

massimizzare la stabilità del software per cercare di ridurre al minimo il numero degli interventi.

7.6 Commercializzazione

Il target che si vuole raggiungere influenza ovviamente le strategie di commercializzazione nonché la pianificazione dello sviluppo del software; il campo in cui DataRiver potrebbe avere maggiore successo, facendo riferimento allo stato attuale di sviluppo, potrebbe essere quello della business intelligence per cui è possibile ipotizzare un utilizzatore con buone capacità tecniche in particolare nell'ambito dei database, nonché una certa dimestichezza con le sorgenti dati aziendali.
Lo scenario ipotizzato è quello di un operatore con competenze tecniche che fornisce analisi e statistiche sui dati al management; in questo caso DataRiver sarebbe in vantaggio rispetto ad una classica soluzione ETL perché non richiede la duplicazione dei dati ne la manutenzione di un nuovo servizio, si aggirerebbe temporaneamente il problema delle performance, nonché quello relativo alla sicurezza perché si può ipotizzare un utilizzo principalmente monoutente, o comunque da un numero molto ristretto di utenti.

Per la commercializzazione di ogni progetto Open Source va deciso inoltre cosa rendere open e cosa tenere come proprietario.
Nel caso di DataRiver la risposta a questo quesito è breve quanto dirompente: tutto.
Il primo motivo è tecnologico: i componenti di DataRiver sono strettamente integrati tra di loro ed è molto difficile disgregarne alcuni senza compromettere radicalmente la funzionalità del sistema: il funzionamento di DataRiver si basa su una serie di passi da seguire in sequenza, toglierne uno o limitarlo compromette l'intero processo. Il query manager, ad esempio, potrebbe essere facilmente tolto dalla release open ma l'utilizzo del sistema sarebbe fortemente compromesso rendendo DataRiver, a livello pratico, di

scarso valore. Poter costruire la GVV senza però poi poterla interrogare risulterebbe parecchio limitante. Inoltre risulterebbe evidente lo scarso interesse di DataRiver per una politica veramente Open Source e difficilmente si riuscirebbe ad attirare nuovi sviluppatori.
I wrapper potrebbero sembrare i componenti logicamente più facili da tenere proprietari, per poterli offrire a pagamento, ma anche questa, benché una via tecnicamente percorribile, è poco proficua. Se DataRiver dovesse incontrare degli ostacoli cercando di affermarsi come nuovo prodotto rendere più difficoltoso il suo utilizzo non migliorerebbe la situazione; se DataRiver al contrario dovesse riscuotere un certo successo, avendo accesso al resto del codice sorgente, i wrapper potrebbero essere riscritti facilmente dalla comunità in qualche mese.
Come risultato finale ci si potrebbe trovare o ad ostacolare la diffusione di DataRiver o ad aver utilizzato il tempo per scrivere due volte gli stessi wrapper invece che scriverne dei nuovi e allargare il supporto a nuove sorgenti.
All'opposto si potrebbe tenere tutto il codice proprietario tranne i wrapper e il minimo indispensabile a cui bisogna avere accesso per poterne scrivere dei nuovi; questo potrebbe stimolare la scrittura di nuovo codice da parte di terzi ma entriamo ampiamente nel free riding (ci sarebbe da chiedersi chi effettivamente regalerebbe del tempo ad una azienda esterna) con in più la ricaduta negativa che un modello proprietario comporta.
Separare SIM o SLIM, cioè togliere al programma la capacità di inferire relazioni intra e inter schema, praticamente azzererebbe il vantaggio tecnologico su gli altri prodotti commerciali; DataRiver sarebbe ridimensionato ad un semplice mappatore manuale come tanti altri, con in più il problema di non avere una interfaccia studiata per facilitare il lavoro di annotazione manuale.
La concorrenza a livello commerciale è già molto agguerrita e difficilmente si potrà affrontare uno scontro contro colossi come IBM e Informatica utilizzando le stesse strategia di mercato; l'Open Source può essere quindi il terreno su cui riuscire effettivamente a competere.

Offrire un prodotto completamente Open Source darebbe oltre a ciò numerosi vantaggi:

- Una maggiore credibilità davanti alla comunità: oltre che stimolare la collaborazione e la libera circolazione di codice e nuove soluzioni, avere un progetto con uno statement chiaro è importante per attirare persone verso la comunità. Avere un atteggiamento ambiguo può vanificare qualsiasi sforzo per creare una community portando ad avere perso i vantaggi del modello proprietario e di quello open in un colpo solo
- Codice migliore: avendo accesso a tutto il codice il processo di revisione paritaria può avere luogo sull'interezza del software, garantendo una qualità e una solidità maggiore del software.
- Assimilazione di codice esterno: non avendo alcun limite è possibile integrare codice di terzi in qualsiasi punto all'interno del software senza preoccuparsi di licenze o altre difficoltà legali
- Supporto: come visto prima il supporto è una voce fondamentale, forse più importante del software stesso; un prodotto completamente open apre scenari di collaborazione con terzi per offrire supporto professionale e di alta qualità dove ce ne sia la necessità. Nel caso ci sia la richiesta di supporto in una zona geografica non coperta o ci siano difficoltà organizzative di altro genere si può sopperire fornendo il supporto tramite terzi che, avendo accesso anch'essi al codice sorgente, può essere comunque di alta qualità
- Trasparenza verso il cliente: un prodotto completamente open è una garanzia per il cliente di trasparenza che è sicuro di quello che compra e che non ci siano risvolti nascosti inaspettati. Mescolare codice open e proprietario può generare un grosso problema comunicativo verso il cliente che può non comprendere appieno quale codice sia rilasciato e quali siano le conseguenze di tale rilascio (si veda vantaggi per il cliente nel capitolo Open Source).

Lo svantaggio di rilasciare il software nella sua interezza è chiaramente che

non si può trarre profitto dalla vendita delle licenze; un atteggiamento ambiguo però, come già detto, è molto rischioso è può decretare il fallimento di un progetto open, è da valutare di conseguenza se un modello esclusivamente proprietario sia sostenibile e al limite optare per questo.

Se si intraprende la strada dell'Open Source, dovrebbe ormai essere più che chiaro che la fonte di guadagno non è più la vendita della licenza (tranne alcuni casi adottando una doppia licenza, si veda il capitolo sull'Open Source), il problema quindi è definire in anticipo quali possano essere le fonti di guadagno.
Essendo DataRiver un software complesso un modello basato sui servizi al cliente sembra quello più calzante: supporto alla prima installazione, assistenza, formazione degli operatori e consulenza sono le voci più immediate su cui puntare, oltre che alla creazione di versioni personalizzate sulle esigenze del cliente.
Ipoteticamente parlando, DataRiver potrebbe essere anche il punto di ingresso per altro software accessorio a pagamento come ottimizzatori o strumenti avanzati per la business intelligence.

8 Analisi dei concorrenti

8.1 Talend Open Studio

Talend Open Studio[v] è un software Open Source per la data integration materializzata, di tipo spiccatamente ETL, sviluppato da una azienda francese, l'omonima Talend.
Talend nasce nel 2002 a Parigi ma il loro prodotto, Talend Open Studio, ha visto la luce solo nella seconda metà del 2006 ed è quindi un software relativamente recente. Come XAware, presentato di seguito, Talend Open Studio (TOS d'ora in avanti) è nato in un ambito puramente aziendale, lontano dal mondo accademico.

Figura 47: Logo di Talend

L'azienda è privata, non è quotata in borsa e finanziata tramite venture capital provenienti da Galileo Partners e AGF Private Equity.
Galileo Partners è una company per la gestione di venture capital nata nel 1989 in Francia come sussidiaria di Worms & Cie, uno dei nomi storici della finanza francese, ed investe in start up e aziende innovative principalmente in Francia solitamente, e come è avvenuto anche per Talend, tramite un finanziamento iniziale che va da 1 a 5 milioni di euro fino ad un massimo di 10 durante la vita dell'azienda.
AGF Private Equity è un'altra azienda francese e, come ci si potrebbe aspettare, opera nel campo degli equity investment, in particolare con investimenti in start up e aziende non quotate.

TOS come detto è Open Source e non guadagna quindi dalla vendita delle licenze; i tre canali di guadagno su cui si basa sono: supporto ai partner che integrano TOS nei loro prodotti, formazione e hosting.
Per quanto riguarda la formazione Talend sta espandendosi con nuove sedi negli Stati Uniti, Germania, Cina e ovviamente Francia.

Talend offre anche soluzioni Software as a Service (SaaS), per cui offre anche hosting a pagamento per le grandi aziende che vogliano esternalizzare questo servizio.

A capo del management di Talend troviamo:
Bertrand Diard, 32 anni, co-fondatore e CEO (chief executive officer, la massima autorità in azienda, è responsabile della gestione della azienda nel suo complesso). Diard è fondatore inoltre della Open Solution Alliance, un consorzio di aziende dedicate allo sviluppo di soluzioni Open Source professionali a cui aderiscono tra gli altri SourceForge e JasperSoft. Prima di entrare in Talend è stato presidente di Brainsoft di Neurones Groupe.

Fabrice Bonan, 34 anni, co-fondatore e COO (Chief Operating Officer, è responsabile dell'operatività giornaliera dell'azienda, risponde direttamente al CEO), proviene anche lui dal mondo della data integration anche se non è stato possibile identificare con certezza i nomi delle aziende per cui ha lavorato.
Cédric Carbone, 29 anni, CTO (chief technical officer, focalizzato sulle problematiche tecnologiche dell'azienda), come il fondatore Diard, prima di approdare a Talend era un manager presso Brainsoft.

Come componenti della board aziendale troviamo Gilles André e ovviamente due rappresentanti degli investitori, Jean-François Galloüin per AGF Private Equity e Francois Duliège per Galileo Partners.
Gilles André, 44 anni, ha iniziato la sua carriera nella finanza e ha poi partecipato a diversi progetti tra cui anche alcuni per la nascita di software house come Leonard s Logic, azienda specializzata nella data integration venduta per circa 21 milioni di dollari a Hummingbird nel '97; attualmente è CEO di Augure, azienda specializzata i soluzioni per la business intelligence.
Hubert Catanese è l'unico membro della Advisory Board e ha il compito di agire da consulente su questioni strategiche per i membri della board.

Talend, anche se opera da poco tempo, ha già all'attivo numerosi clienti di alto livello come si vede nella figura sottostante:

Figura 48: Alcuni dei clienti Talent

TOS è uno strumento per la data integration molto utile in campo ETL per la mappatura visuale e semi automatica di più sorgenti in una.
La mappatura avviene in maniera grafica attraverso una serie di wizard molto semplici; la trasformazione dei dati segue un modello a black box collegate tra loro.
Ogni black box manipola i dati in una determinata maniera, collegandone diverse tra loro è possibile ottenere l'output desiderato in uno dei formati supportati.
Una volta che si è costruito il flusso dei dati è possibile eseguire l'effettiva elaborazione dei dati; basandosi sul diagramma costruito graficamente TOS crea del codice Java o Perl che viene poi effettivamente eseguito.
E' inoltre possibile dall'interfaccia di TOS modificare manualmente il codice

generato e aggiungere nuovo codice tramite pacchetti jar.
TOS è in grado di leggere da quasi tutte le sorgenti dati più comuni e produrre un output in una enorme varietà di formati, tra cui:

- CentricCRM
- Microsoft AX
- Salesforce
- SugarCRM
- Jasper
- AS400
- MS Access
- Database tramite JDBC
- DB2
- FireBird
- Ingres
- LDAP
- SQL Server
- MySQL
- Oracle
- PostgreSQL
- FTP
- File di testo
- CSV
- RSS
- XML

L'installazione non verrà trattata, essendo di un software scritto in Java è molto semplice e richiede solo di seguire passo passo il wizard assegnando i vari path ecc.

8.1.1 Utilizzo

Come si vede chiaramente dall'interfaccia, TOS è basato su Eclipse e offre all'utente la classica struttura divisa in pannelli mobili.

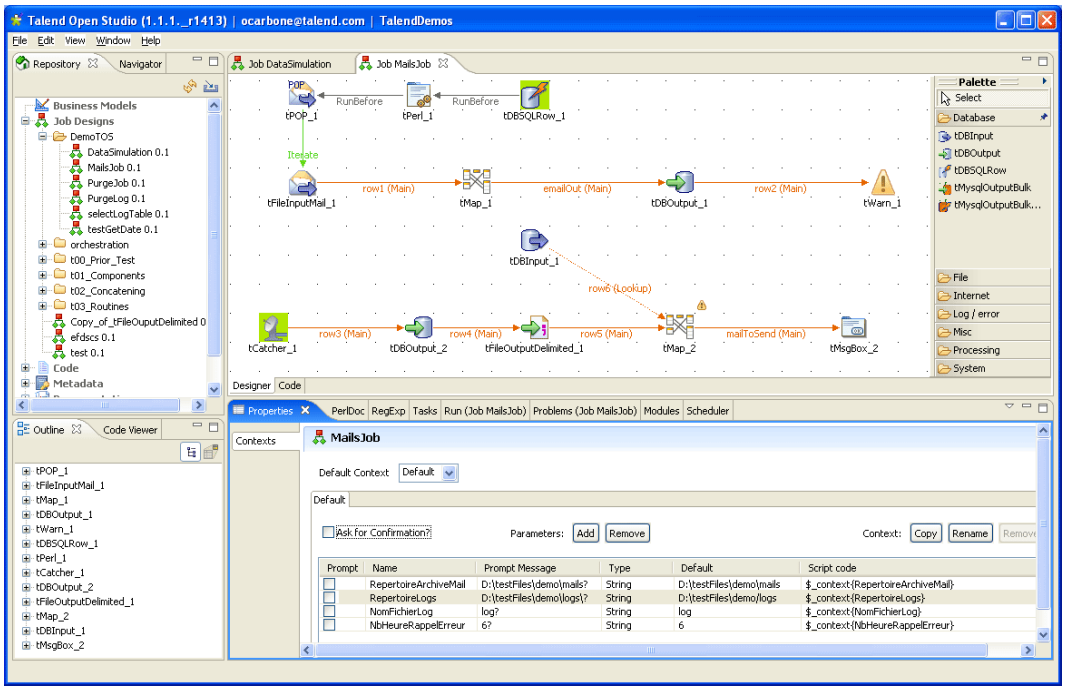


Figura 49: L'interfaccia di Talend

In alto troviamo le toolbar e i menu; nella colonna di sinistra il pannello dei repository che mostra i progetti e i file creati; al centro l'area per il design con affianco la palette di strumenti per la progettazione della nostra applicazione ; sotto l'area a tab per la configurazione dei componenti.

Attraverso la colonna di sinistra sono visibili i repository che contengono l'elenco delle attività possibili; entrando nel dettaglio troviamo i Business Model, una rappresentazione grafica informale delle operazioni o del progetto che si vuole portare a termine.
Creando un nuovo Business Model la palette mostra dei nuovi strumenti che si possono utilizzare:

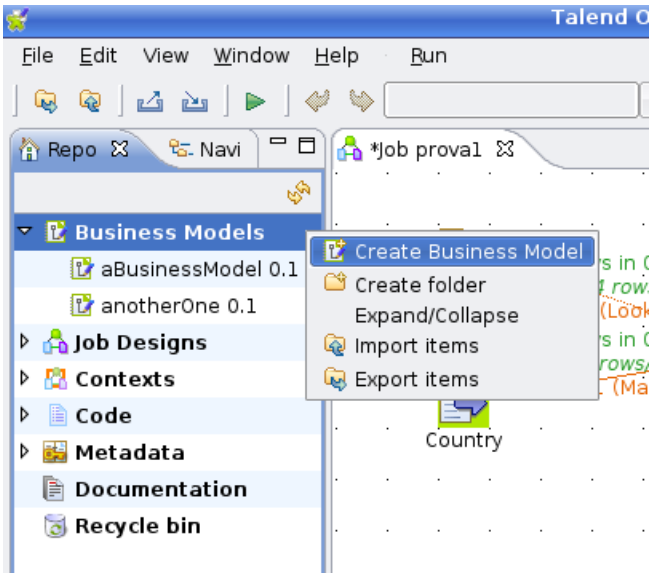


Figura 50: Creazione di un Business Model

Creato il nuovo Business Model si può procedere a disegnare lo schema come si preferisce, esso è puramente indicativo e non vincolante ma è utile per mantenere uno schema del progetto.

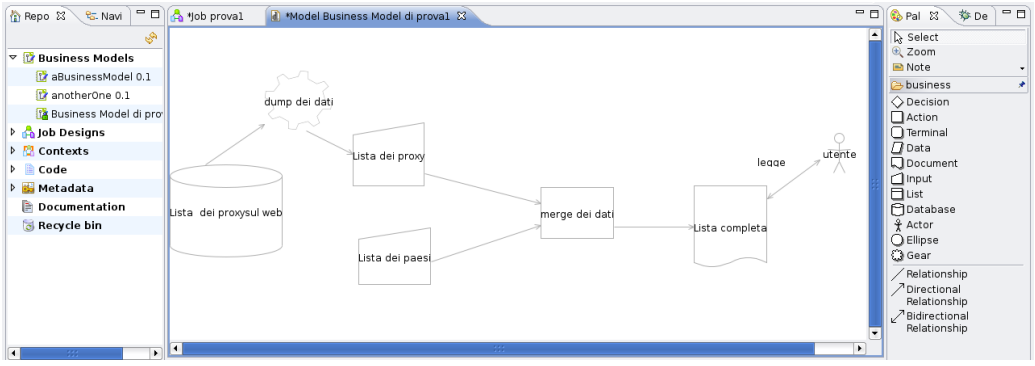


Figura 51: Vista di un Business Model

Lo schema di utilizzo è visuale e basato sul drag&drop; l'idea di principio è quella di prendere dalla palette gli strumenti che servono e dropparli nell'area di design, configurarli tramite il pannello a tab sottostante e poi combinarli tramite delle linee che rappresentano il flusso ideale di

informazioni.

Il Job Design è il cuore del sistema e sarà analizzato in dettaglio poco più avanti.
In Context vengono conservati tutti i file che contengono informazioni di rilievo come i path e altre variabili riguardanti il contesto.
Attraverso Code è possibile manipolare il codice e le routine.
In Metadata vengono conservati tutti i dati importanti che si possono voler riusare come gli schema, i mapping o altri metadati.
E' inoltre possibile agganciare agli oggetti creati all'interno dei vari repository dei file esterni da usare come documentazione, come ad esempio file PDF o OpenOffice.

Vediamo in dettaglio la creazione di un Job attraverso un semplice esempio in cui mappiamo due file Comma Separated Value (CSV) in un nuovo file XML:
il file proxy.csv contiene una lista di open proxy i cui parametri sono: l'ip del proxy, la porta su cui opera, il codice del paese in cui risiede e altri parametri accessori;
il file country.csv contiene la lista dei codici dei paesi e il nome completo corrispettivo.
Il risultato che si vuole ottenere è un nuovo file XML dove per ogni proxy vengano indicati tutti i valori e il nome del paese in cui risiede invece del codice numerico.

Lista di Open Proxy:

```
ip;port;type;ssl;country_code;latency(msec);uptime(per cent);checked(UTC)
201.30.20.176;80;Transparent;false;2;15038;29;"Feb 15 20:53:30"
209.106.223.205;80;Transparent;true;13;3402;36;"Feb 16 12:44:37"
209.106.223.216;80;Transparent;true;13;318;20;"Feb 16 12:48:16"
66.244.214.230;80;Transparent;true;2;15035;27;"Feb 16 12:50:21"
201.15.122.235;80;Transparent;false;1;9875;65;"Feb 16 12:53:08"
63.238.76.250;80;Transparent;false;13;261;54;"Feb 16 12:58:34"
88.191.25.38;80;Transparent;false;4;1446;79;"Feb 16 12:59:46"
196.203.15.40;80;Transparent;false;11;28009;43;"Feb 16 13:01:44"
202.134.73.55;80;Transparent;false;7;16322;38;"Feb 16 13:05:48"
95.229.236.106;80;Transparent;false;12;1350;97;"Feb 16 13:13:08"
85.25.140.160;80;Transparent;false;5;42238;25;"Feb 16 13:15:43"
200.65.127.161;80;Transparent;false;10;140;99;"Feb 16 13:16:06"
63.228.245.127;80;Distorting;false;13;26020;7;"Feb 16 13:17:00"
125.162.85.174;80;Transparent;false;9;3122;31;"Feb 16 13:19:07"
74.86.12.82;80;Transparent;false;13;493;70;"Feb 16 13:29:01"
85.216.21.93;80;Transparent;false;5;22167;31;"Feb 16 13:29:28"
62.6.242.14;80;Transparent;true;6;12425;50;"Feb 16 13:34:44"
194.0.163.148;80;Transparent;false;5;10709;50;"Feb 16 13:34:55"
196.27.107.158;80;Transparent;true;14;7171;50;"Feb 16 13:35:27"
201.229.208.2;80;Transparent;false;3;1859;98;"Feb 16 13:38:19"
210.211.149.200;80;Transparent;false;8;2064;50;"Feb 16 13:38:42"
148.233.159.58;80;Transparent;true;10;185;50;"Feb 16 13:42:42"
202.84.17.42;80;Transparent;true;7;5293;84;"Feb 16 13:43:43"
165.228.128.11;80;Transparent;false;0;893;69;"Feb 16 14:01”

      [...]
```

Lista contenente i country code:

```
country; country_code
"Australia";0
"Brazil”; 1
"Canada";2
"Dominican Republic";3
"France";4
"Germany";5
"Great Britain (UK)";6
"Hong Kong";7
"India";8
"Indonesia";9
"Mexico";10
"Tunisia";11
"United Arab Emirates";12
"United States";13
"Zimbabwe";14

      [...]
```

La costruzione del mapping è visuale in Talend, quindi prendiamo dalla palette l'icona del file input CSV e la droppiamo nell'area di lavoro:

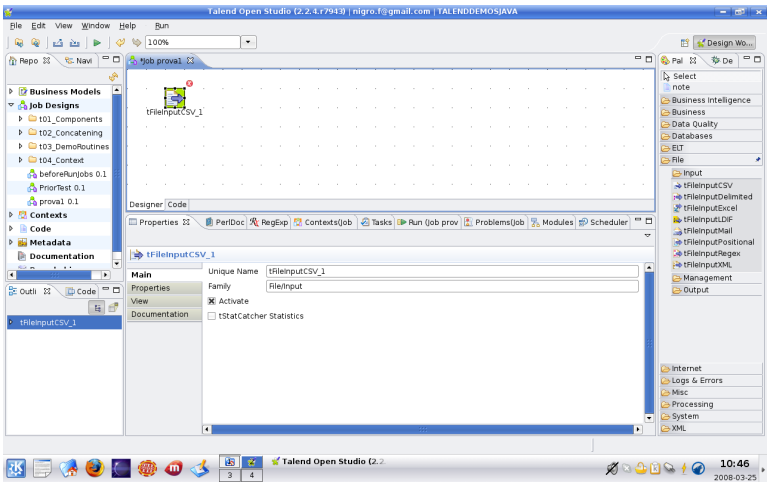


Figura 52: Inserimento di una sorgente

Assegniamo all'icona una etichetta, inseriamo il path per il file di input e il separatore utilizzato, in questo caso ';' .

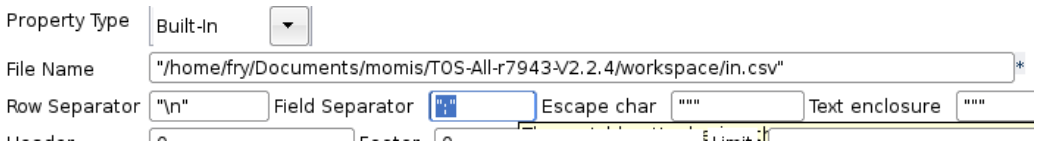


Figura 53: Definizione del separatore

Definiamo lo schema dei valori che contiene il file CSV:

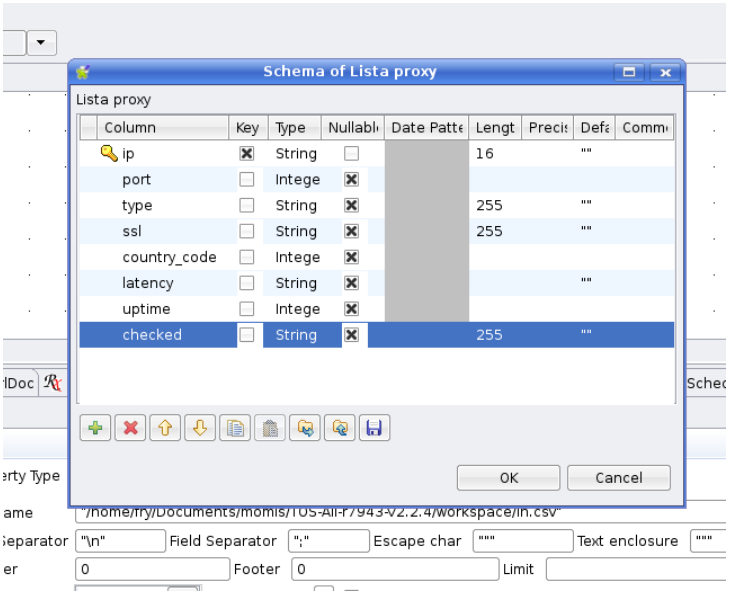


Figura 54: Definizione dei campi della lista dei proxy

Eseguiamo le stesse operazioni country.csv:

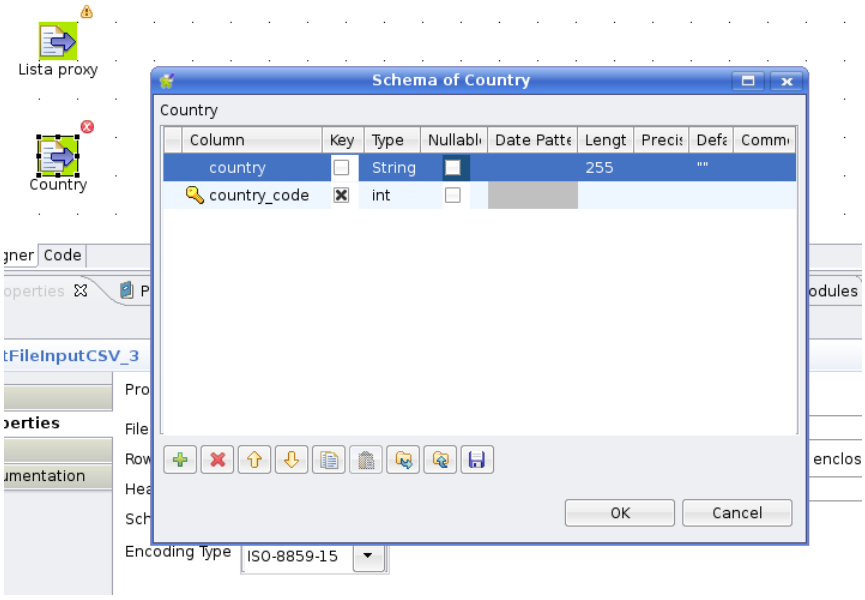


Figura 55: Definizione dei campi della lista dei country code

Dall'area Processing della palette prendiamo l'icona tMap, la droppiamo nell'area di lavoro e ci colleghiamo i due file inseriti prima:

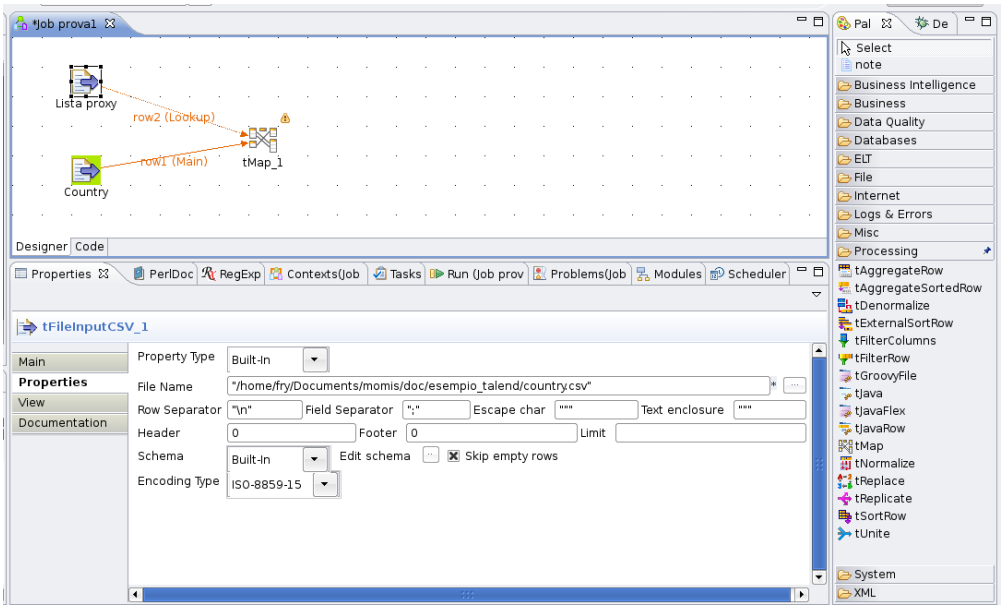


Figura 56: Definizione del mapping

Inseriamo l'icona per il file di output XML e la colleghiamo con l'icona di mapping:

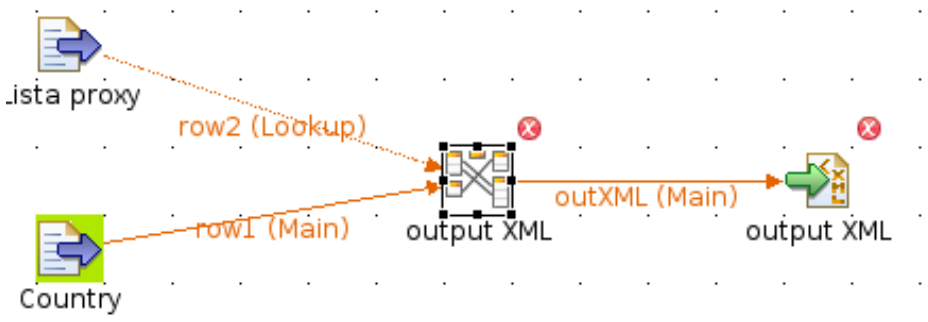
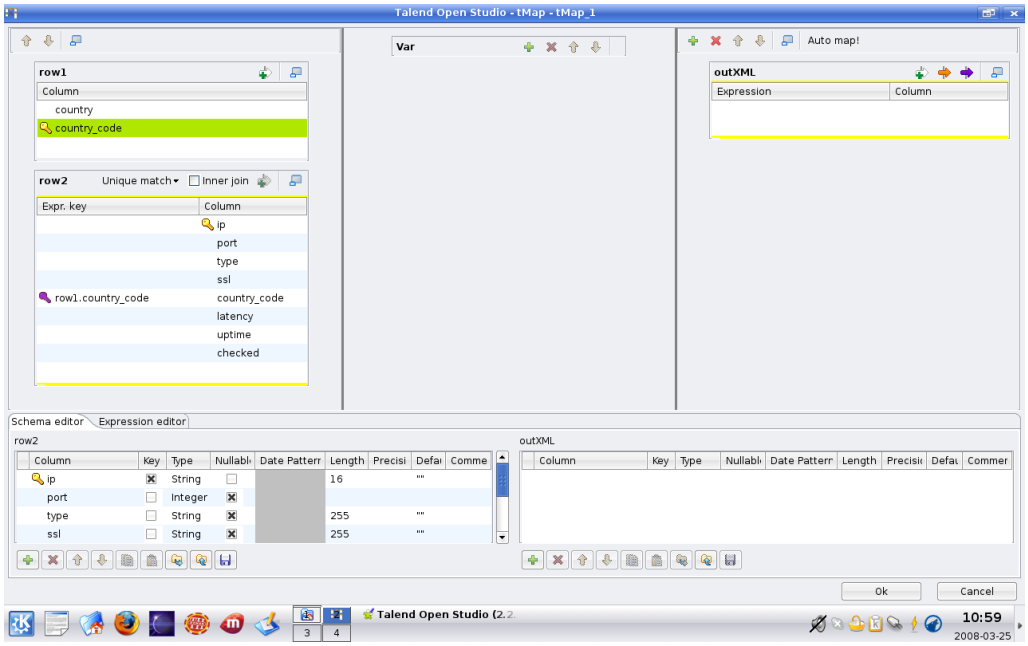


Figura 57: Definizione dell'output

A questo punto dobbiamo eseguire il mapping vero e proprio tra i due file

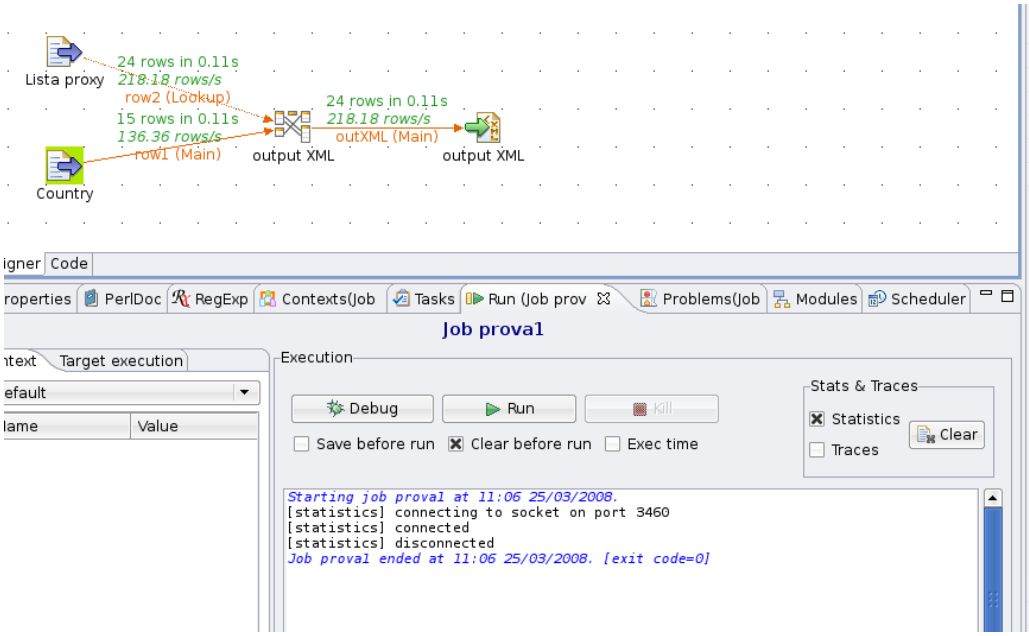
sorgente, apriamo quindi lo schema editor.
Nella colonna di sinistra vediamo lo schema dei file di input e in quella di destra quello del file di output; il file di output potrebbe in realtà essere un qualsiasi altro formato, un database ecc.
Nella colonna centrale, adesso vuota, è possibile inserire delle tabelle intermedie per la manipolazione dei dati.



Per prima cosa va assegnato un join tra il country_code del file country.csv e il country_code del file proxy.csv; come si vede nella figura la riga è segnata in verde per country.csv e compare la chiave viola in proxy.csv.

A questo punto è possibile trascinare i campi che vogliamo avere in uscita e dropparli nella colonna di destra:

Il mapping è concluso, chiudiamo lo schema editor ed eseguiamo il job:



Come si vede è possibile attivare le statistiche ed avere in tempo reale la situazione delle operazioni che vengono compiute.

File di output XML:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<row>
<ip>165.228.128.11</ip>
<port>80</port>
<type>Transparent</type>
<ssl>false</ssl>
<latency>893</latency>
<uptime>69</uptime>
<checked>Feb 16 14:01</checked>
<country>Australia</country>
</row>
<row>
<ip>201.15.122.235</ip>
<port>80</port>
<type>Transparent</type>
<ssl>false</ssl>
<latency>9875</latency>
<uptime>65</uptime>
<checked>Feb 16 12:53:08</checked>
<country>Brazil</country>
</row>
<row>
<ip>201.30.20.176</ip>
<port>80</port>
<type>Transparent</type>
<ssl>false</ssl>
<latency>15038</latency>
<uptime>29</uptime>
<checked>Feb 15 20:53:30</checked>
<country>Canada</country>
</row>

      [...]
```

Adesso è possibile cambiare l'output senza troppa difficoltà in uno qualsiasi dei formati supportati semplicemente cambiando l'icona del file di uscita e rilanciando il job (dopo essersi accertati che il mapping vada ancora bene...). Ovviamente è possibile complicare l'esempio utilizzando più file di output di formati diversi, oppure utilizzando ad esempio un database dove memorizzare i messaggi di errore o i log e così via. In questo caso si sostituisce semplicemente il file XML di output con un file CSV, e si controlla il mapping e il job viene rilanciato:

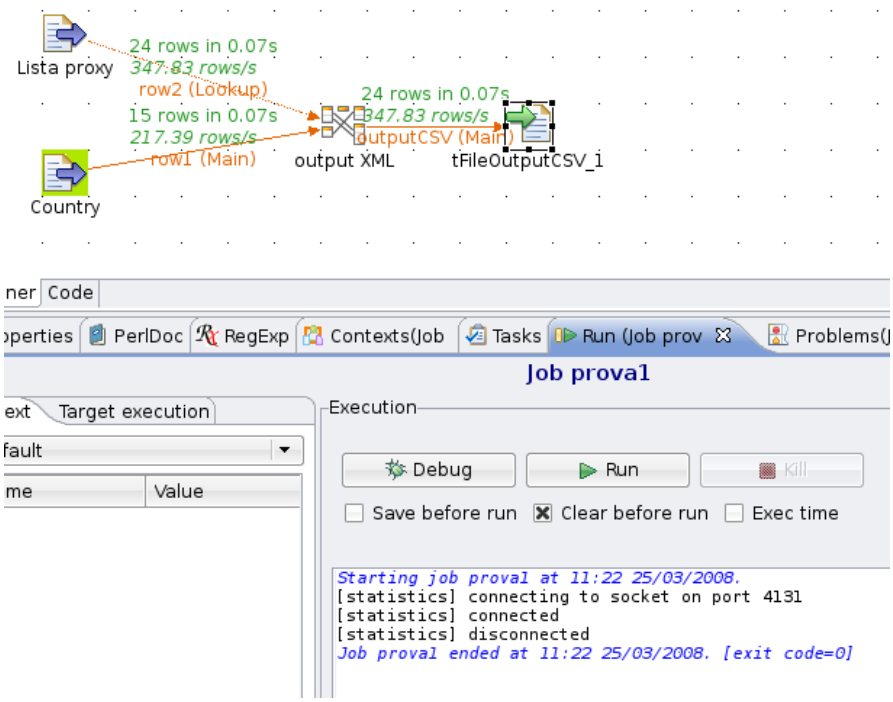


Figura 58: Esecuzione con le statistiche attivate

Output su file CSV:

```
"165.228.128.11";"80";"Transparent";"false";"893";"69";"Feb 16
14:01";"Australia"
"201.15.122.235";"80";"Transparent";"false";"9875";"65";"Feb 16
12:53:08";"Brazil"
"201.30.20.176";"80";"Transparent";"false";"15038";"29";"Feb 15
20:53:30";"Canada"
"66.244.214.230";"80";"Transparent";"true";"15035";"27";"Feb 16
12:50:21";"Canada"
"201.229.208.2";"80";"Transparent";"false";"1859";"98";"Feb 16
13:38:19";"Dominican Republic"
"88.191.25.38";"80";"Transparent";"false";"1446";"79";"Feb 16
12:59:46";"France"
"85.25.140.160";"80";"Transparent";"false";"42238";"25";"Feb 16
13:15:43";"Germany"
"85.216.21.93";"80";"Transparent";"false";"22167";"31";"Feb 16
13:29:28";"Germany"
"194.0.163.148";"80";"Transparent";"false";"10709";"50";"Feb 16
13:34:55";"Germany"
"62.6.242.14";"80";"Transparent";"true";"12425";"50";"Feb 16
13:34:44";"Great Britain (UK)"
"202.134.73.55";"80";"Transparent";"false";"16322";"38";"Feb 16
13:05:48";"Hong Kong"

      [...]
```

8.2 XAWARE

XAware Inc.[b] è un'azienda americana con sede a Colorado Springs e una filiale europea in Svizzera; l'azienda ha incominciato le proprie attività nel 1997 come una tradizionale software house incentrata su prodotti proprietari per la data integration, modello economico che ha seguito per dieci anni, fino alla fine del 2007; la sua nascita è legata ad un contesto puramente aziendale, lontano da ambienti accademici o istituzionali.

Figura 59: Logo di XAware

Tim Harvey, CEO dell'azienda, laureato in finanza presso l'università della Florida, ha servito per quattro anni nel corpo dei Marines prima di entrare in Datalogix, che sviluppava software per l'automazione industriale, acquisita nel '96 da Oracle. Prima dell'ingresso in XAware ha sostenuto diversi incarichi tra cui è stato Senior Vice President del reparto Vendite e Marketing di S1[o], azienda americana specializzata nel software finanziario e di servizi di pagamento.

Sean Dwyer, Senior Vice President del reparto vendite e marketing, come il CEO Tim Harvey, ha precedentemente lavorato come general manager per S1 e prima ancora per BEA, ora in via di acquisizione da parte di Oracle.

Kirstan Vandersluis, Chief Scientist (una figura simile al CTO), è stato il fondatore di XAware ed ha anche scritto parte del software; specializzato nello sviluppo software ha lavorato precedentemente per varie aziende di diversi ambiti: servizi finanziari, banche, assicurazioni e telecomunicazioni tra cui MCI, importante azienda di telecomunicazioni americana acquisita di recente da Verizon.

XAware Inc. è entrata solo recentemente nel mercato Open Source, precisamente dal novembre del 2007, data in cui ha rilasciato, con ampio

successo visto i 60.000 e più download effettuati in meno di sei mesi, XAware sotto licenza GPLv2 come scelta strategica per la persecuzione della mission aziendale:

> **XAware s mission is to be the world s most popular way of integrating data and applications.**
>
> Tim Harvey, XAware CEO

Con il rilascio sono arrivati anche nuovi investimenti in venture capital per circa 7,4 milioni di dollari da parte di una cordata composta da Sequel Venture Partners, ITU Ventures e BMJP e GMT; i primi tre investitori hanno già contribuito a finanziare l'azienda fin dal 2002 con una somma iniziale di 2,2 milioni di dollari diventata poi nel tempo circa 18 milioni di dollari. Ad oggi gli investitori possiedono la maggioranza in azienda mentre il resto è detenuto dagli impiegati e dal management.
Il modello di business di XAware si basa principalmente sull'offerta di servizi di supporto, consulenza e training; inoltre l'offerta è arricchita da alcuni software accessori a pagamento studiati per gli ambienti enterprise, per il management avanzato e il reporting.

XAware Inc. impiega una cinquantina di persone per lo sviluppo del suo prodotto principale per la data integration, l'omonimo XAware, che genera un volume d'affari di poco più di 4 milioni di dollari l'anno (dati riportati da Hoovers[c]).
In concomitanza con il lancio del prodotto è stato aperto un secondo portale aziendale, www.xaware.org, orientato all'aggregazione, gestione e promozione della comunità che gravita intorno alla nuova versione open; tra le FAQ del nuovo portale si trova anche questa eloquente risposta che fuga ogni ulteriore dubbio sulla scelta di XAware Inc. di rilasciare il proprio prodotto:

> **Why has XAware, Inc. elected to make its industry leading Data Integration software available to the Open Source**

Community in the form of the XAware 5 Open Source Project?

> Open source has proven to be a powerful model for driving the rate of innovation and productive use of software through the opening and sharing of source code, engaging in collaborative development with a community of talented open source developers worldwide, partnering with other open source projects, and participating in the open source ecosystem. Infrastructure software and tools in general, and data integration software in particular, are well suited for the open community innovation and participation. XAware, Inc., recognizing the value the community can bring in driving innovation and use of its software, determined that open source was the best way to pursue its mission of becoming the world s most popular way of integrating data and applications.

I clienti di XAware Inc. sono diversi ma si concentrano in particolare nei campi delle assicurazioni e della finanza.

Figura 60: Alcuni dei clienti di XAware

In particolare il caso AXA ha suscitato parecchio interesse da parte degli

investitori, visto che che AXA ha un fatturato nell'ordine di 100 miliardi di dollari annui. XAware ha fornito l'infrastruttura che permette ai broker di AXA di eseguire l'upload in tempo reale delle nuove polizze avendo un feed back immediato e, successivamente, è stato implementato anche un sistema per la creazione di preventivi in real time.

Fino ad ora XAware Inc. ha seguito un modello di sviluppo legato alla vendita della licenza dei propri prodotti software ma come tante altre aziende che operano nel mercato Open Source questo modello ha dovuto necessariamente subire delle modifiche per spostarsi più sul lato dei servizi. Al momento è possibile utilizzare XAware sotto i termini XAware Desktop License o XAware Enterprise License: come si può immaginare la Desktop License è completamente libera, si può scaricare il software gratuitamente, ma non viene fornito alcun tipo di supporto.
L' Enterprise License fornisce al contrario il supporto e tutti i servizi che normalmente vengono richiesti dal mondo aziendale come: consulenza, assistenza telefonica e on-site, supporto al deployment ecc.
Oltre a ciò, sempre a pagamento, sono disponibili anche tutti i servizi accessori ma fondamentali come la formazione tramite corsi, certificazioni e la manualistica.
La complessità del prodotto e dell'ambiente di utilizzo, nonché la criticità delle informazione che gestisce XAware, supportano ottimamente un modello economico incentrato sui servizi e sul supporto portando a XAware un discreto successo.

8.2.1 Principi di funzionamento

Prima di incominciare va sottolineato che la nuova release Open Source, la 5.0, è meno ricca di funzionalità della precedente 4.5, completamente proprietaria; nella versione open infatti sono stati eliminati alcuni moduli che, per via della licenza con cui sono rilasciati, non possono essere resi

open. I moduli che sono stati eliminati sono in via di riscrittura e verranno reintegrati nella versioni future sotto GPL.

XAware è un prodotto per la data integration molto flessibile e può essere impiegato sia come classico strumento ETL quanto per la data federation. Semplificando, XAware permette di accedere a diverse sorgenti dati in real time attraverso una view XML che mappa diverse sorgenti in una rappresentazione globale e unificata, che può poi essere messa a disposizione attraverso dei web service, application server o anche integrata all'interno di altre applicazioni.
Essendo tutti i dati presentati in XML la visualizzazione può essere delegata anche ad altri programmi esterni a XAware, come ad esempio un web server.

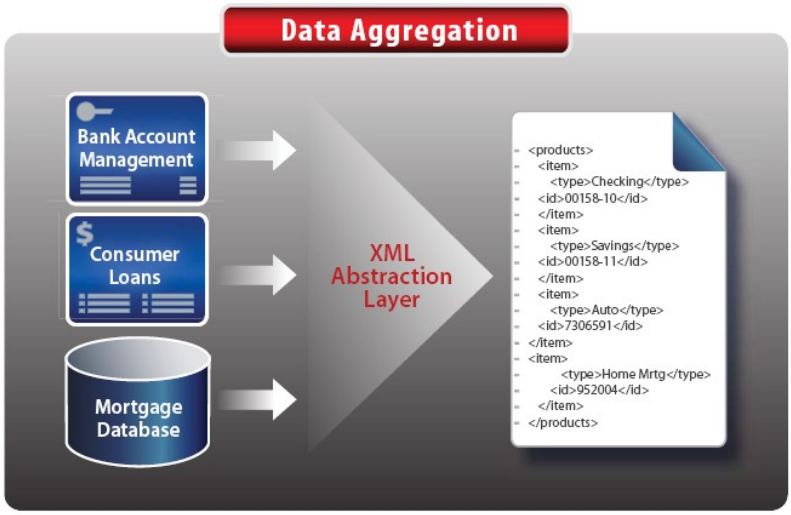


Figura 61: Data Aggregation

In pratica una view XML è una raccolta di di documenti XML che contengono tutti i meta-dati sulle operazioni necessarie per accedere ai dati, manipolarli e consegnarli all'esterno; attraverso una view è inoltre possibile scatenare processi esterni o richiamare dei Javascript o del codice Java custom.
Vediamo brevemente i principali scenari di utilizzo in cui è impiagato XAware: lo scenario di utilizzo più classico è senz'altro la **Data Aggregation**, in cui diverse sorgenti vengono mappate in una view globale che permette di utilizzare tutte le sorgenti come fossero una unica, permettendo alle

applicazioni di non doversi preoccupare di come reperire i dati. La caratteristica fondamentale della Data Aggregation è l'indipendenza delle diverse sorgenti, permettendo di eseguire quindi le query in parallelo e in maniera autonoma, permettendo inoltre di ottimizzare il sistema.

Il **Data Chaining** può essere visto come una variazione della Data Aggregation, dove la differenza sostanziale risiede nella interdipendenza delle sorgenti; una query, per essere eseguita nella sua interezza, deve seguire un ordine ben preciso di interrogazione delle varia sorgenti. I dati

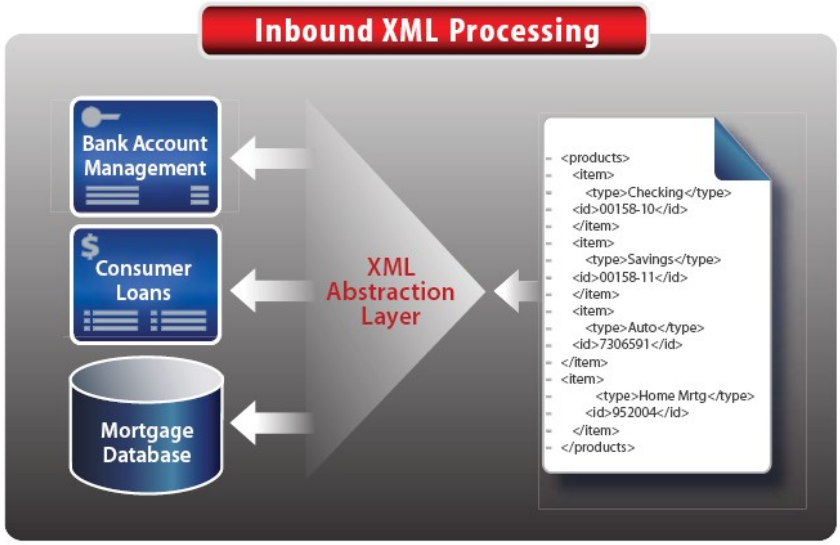


Figura 62: Inbound Processing

ottenuti da una sorgente sono indispensabili per poter eseguire la ricerca nella sorgente successiva.

Sia la Data Aggregation che il Data Chaining si concentrano sulla lettura delle sorgenti ma, con XAware, è anche possibile scriverci, attraverso l'**Inbound Processing.** Dall'esterno è possibile, sempre tramite XML, eseguire anche delle scritture nella view globale oltre che letture; sarà XAware poi che si occuperà di scomporre la scrittura e inviare i dati alle diverse sorgenti. Un aspetto delicato di cui tenere conto in questo caso è la gestione delle transazioni ed eventuali rollback: XAware è in grado di gestire transazioni atomiche solo su sorgenti che siano già in grado di gestirle singolarmente; se

la sorgente dati non è un database, ma ad esempio un file excel, la gestione delle transazioni deve essere implementata manualmente, cosa per altro non banale.

Nel caso ci sia la necessità di mettere in comunicazione due entità, che possono essere due aziende diverse, due filiali o altro, è possibile usare XAware come piattaforma comune intermedia per l'**Information Exchange**. In questo caso XAware opera solo da convertitore per trasformare i dati dal formato usato da un soggetto nel formato utilizzabile dall'altro, passando attraverso uno stadio comune in XML. Nel caso i dati provenienti dalle due aziende siano già in XML ma con in due formati differenti viene generato semplicemente un XSL per trasformare un formato XML nell'altro e viceversa.

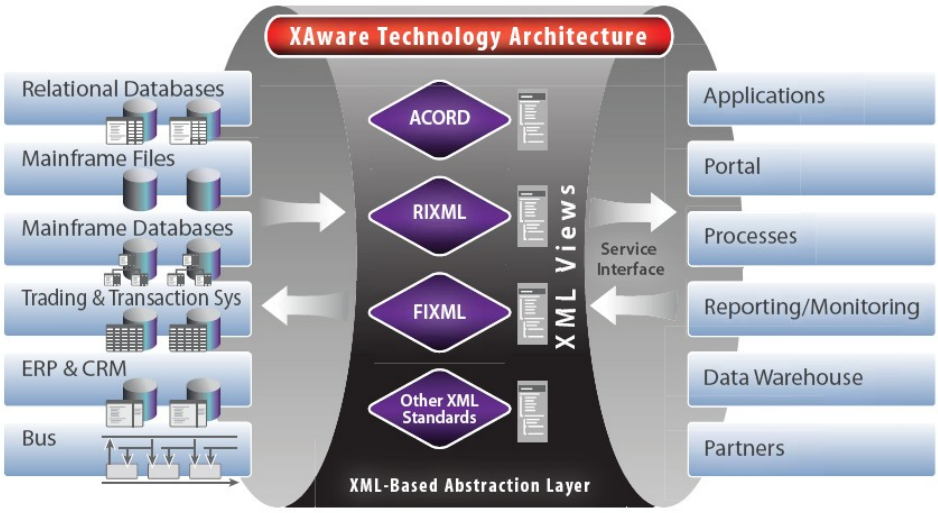


Figura 63: Vista ad alto livello dell'architettura di XAware

Tutte le interazioni tra la view e il mondo esterno vengono effettuate anch'esse attraverso XML: il sistema mostra all'esterno dati in XML e dall'esterno si scrivono dati nella view in XML; ciò garantisce un livello di interoperabilità elevato e basato su standard.

Come si vede possono essere adottati numerosi schemi XML standard per le view, anche a seconda del campo di utilizzo. In figura vediamo alcuni standard utilizzati come ACORD per le assicurazioni, RIXML per gli investimenti finanziari e FIXML per lo scambio di informazioni finanziarie in

generale.
La view può essere quindi cambiata o espansa, modificando o aggiungendo nuove sorgenti dati, senza che le applicazioni all'esterno se ne accorgano.
Inoltre la gestione degli accessi e della sicurezza è spostata dalla sorgenti dati, che devono essere tutte accessibili da XAware, nella view, che si occupa di garantire e controllare le autenticazioni, gli accessi e la cifratura dei dati; ciò avviene secondo uno schema role-based tramite ACL conforme alle specifiche Java Authentication and Authorization Service (JAAS) e che può quindi anche essere integrato con un sistema LDAP o con altri single-sign-on frameworks.
Volendo si può gestire la sicurezza ad un livello ancora più raffinato stabilendo la visibilità dei singoli data set, così da poter stabilire quali dati possono essere aggregati e visti.
Nella nomenclatura ufficiale utilizzata all'interno di XAware ogni cosa aggiunge il suffisso Biz ; pertanto una view XML diventa una BizView e così via.
Andando un po' in profondità una BizView risulta essere formata da tre componenti: un BizDocument, vari BizComponent e vari BizDriver; tutti questi componenti sono in ultima analisi dei file XML.

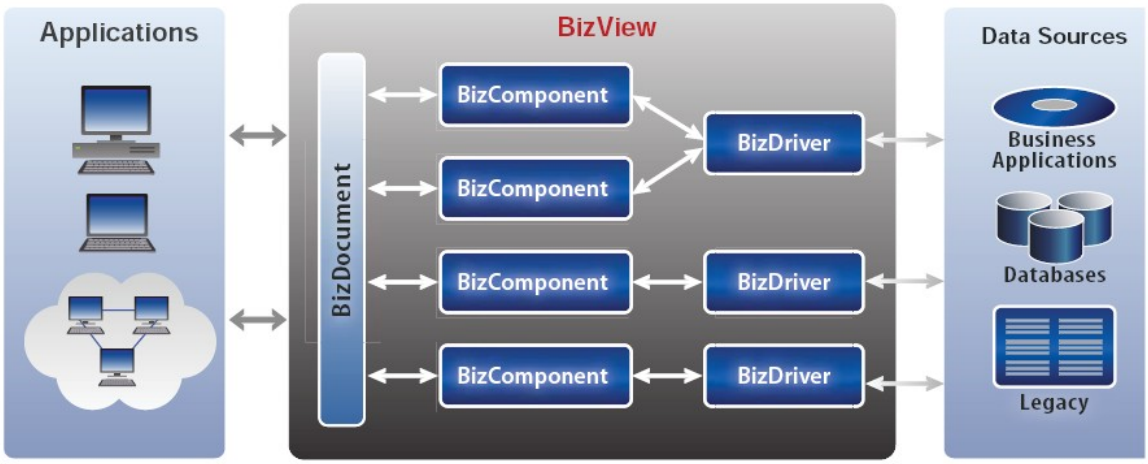


Figura 64: Architettura di una BizView

Il BizDocument è il documento più importante della BizView e contiene le informazioni riguardanti il formato XML utilizzato per la rappresentazione dei dati, i passi necessari per processare e trasformare i dati e i riferimenti ai BizComponent.

Esempio di BizDocument:

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrderList
                    xmlns:xa="http://xaware.org/xas/ns1"
                    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                    xa:input_type="PurchaseOrderList"
                    xa:on_error="xa-doc::/PurchaseOrderList/Error"
                    xsi:noNamespaceSchemaLocation="../schema/POList.xsd"
                    xa:transaction="required">
      <xa:input />
      <PurchaseOrder xa:bizcomp="orders/insertOrders1.xbc"
                    xa:input="xa-input::/PurchaseOrderList/PurchaseOrder"
                    xa:remove="yes" />
      <Error xa:include="no">
                    <detail>$xavar:error$</detail>
      </Error>
</PurchaseOrderList>
```

Come si può vedere un BizDocument è un semplice file XML; le direttive che iniziano con xa: fanno parte del namespace specifico definito all'interno di XAware dalla direttiva xmlns.

Un BizComponent contiene le informazioni necessarie per leggere, trasformare e formattare i dati prelevati dalle sorgenti in XML e viceversa; in pratica un BizComponent è visto dal BizDocument come un insieme di data set generati dinamicamente. E' possibile rendere un BizDocument riutilizzabile parametrizzandone l'esecuzione, cioè passandogli all'avvio, appunto, dei parametri o un intero file XML di configurazione.

Un BizComponent è il principale blocco di esecuzione di una BizView ed è generalmente sempre composto dalle seguenti parti:

- Input, specificato attraverso xa:input
- la richiesta, definita da xa:request
- il tipo degli elementi delle risposta, definito da xa:return
- la struttura dei dati della risposta, definita da xa:response

Esempio di un BizComponent:

```
<?xml version="1.0" encoding="UTF-8"?>
<FILETEST xmlns:xa="http://xaware.org/xas/ns1" xa:bizdriver="myCompany/
common/fileRead.xdr" xa:filetoRead="%filetoRead%" xa:bizcomptype="
FILE" xa:response_type="getManagers">
      <xa:input>
            <xa:param xa:name="filetoRead" xa:datatype="string"
xa:default="
            data/managers.txt" xa:description="File to read" />
      </xa:input>
      <xa:description />
      <xa:request xa:field_separator="|" />
      <xa:response xa:set_text="xa-current::./managerList/manager/phone"
      xa:set_text_value="888-999-1212">
            <managerList>
                  <manager id="%1%">
                        <name>%4% %3%</name>
                        <phone>%2%</phone>
                        <dept>
                              <no>%5%</no>
                              <name>%6%</name>
                              <loc>%7%</loc>
                        </dept>
                  </manager>
            </managerList>
      </xa:response>
</FILETEST>
```

Un BizDriver, infine, contiene le istruzioni strettamente necessarie per collegarsi ad una sorgente come, ad esempio, la stringa di connessione ad un database; i BizDriver non sono strettamente necessari al funzionamento dei BizComponent, infatti, nel caso in cui la connessione avvenga attraverso metodi standard, come HTTP o FTP ad esempio, i BizComponent possono accedere alle sorgenti direttamente.
Nella maggior parte dei casi tuttavia è necessario creare un BizDriver per ogni differente sorgente dati che si utilizza, anche se è poi possibile

riutilizzarlo per più BizComponent; XAware fornisce già una serie di BizDriver preassemblati che possono essere personalizzati a piacimento.

Esempio di un BizDriver:

```
<?xml version="1.0" encoding="UTF-8"?>
<xa:bizDriver xmlns:xa="http://xaware.org/xas/ns1" xa:bizdrivertype="
SQL_JDBC" xa:version="5.0">
      <xa:description>Connects to the account representatives database</
      xa:description>
      <xa:poolingparams />
      <xa:connection>
            <xa:url>jdbc:derby://localhost:1527/accounts</xa:url>
            <xa:user>me</xa:user>
            <xa:pwd>mine</xa:pwd>
            <xa:factory>org.apache.derby.jdbc.ClientDriver</xa:factory>
            <xa:properties />
      </xa:connection>
</xa:bizDriver>
```

Come si vede anche in questo caso il fatto che si tratti di file XML rende molto semplice la lettura e la comprensione del contenuto.

I BizComponent costituiscono la componente dinamica di una BizView e comunicano da un lato con il BizDocument, che li mette in esecuzione, fornendogli i dati trasformati in XML e nascondendo al tempo stesso tutti i dettagli e la complessità comunicativa di accesso alle sorgenti, dall'altro con le sorgenti tramite, se necessario, i BizDriver.
Un BizComponent restituisce sempre dei dati XML su cui il BizDocument può operare in maniera uniforme, qualsiasi sia la sorgente dati in questione.
Il BizDocument opera quindi solo su dati XML e per integrare, ad esempio, dati provenienti da un RDBMS e da un mainframe, il BizDocument innescherà l'esecuzione dei relativi BizComponent che restituiranno dati XML; sarà poi il BizDocument a integrare i dati attraverso join XML, modifiche della gerarchia

degli XML e così via.
Un BizComponent può a sua volta mandare in esecuzione altri BizComponent o anche altri BizDocument che possono innescare altri BizComponent in maniera ricorsiva; alla fine dell'esecuzione vengono comunque restituiti al BizDocument originale dei dati in XML che è in grado di manipolare a piacimento e mostrare quindi all'esterno.
Una volta definito, un BizComponent può essere riutilizzato e invocato da più BizDocument.

8.2.2 Componenti

XAware è composto da quattro componenti principali:
il Designer, un programma Java basato su Eclipse che assiste l'integration designer nella creazione delle BizView tramite un approccio semi-visuale mescolando un'interfaccia drag&drop alla programmazione Java, se necessario.
L'engine si occupa effettivamente dell'integrazione gestisce le richieste degli utenti.
Gli adapters fanno da interfaccia tra l'engine e le sorgenti dati che possono essere RDBMS, LDAP, SAP, Excel e altre e vengono implementati tramite BizComponent e BizDriver;
I connectors infine rappresentano invece il punto di accesso dei client che possono utilizzare HTTP, SOAP o altro.

L' Engine (XA-Designer) è l'environment responsabile di far funzionare XAware, mettere in esecuzione i vari componenti, gestire le richieste e così via.
XA-Engine può funzionare in tre diverse modalità:

nella configurazione stand-alone XA-Engine è impacchettato in un jar che

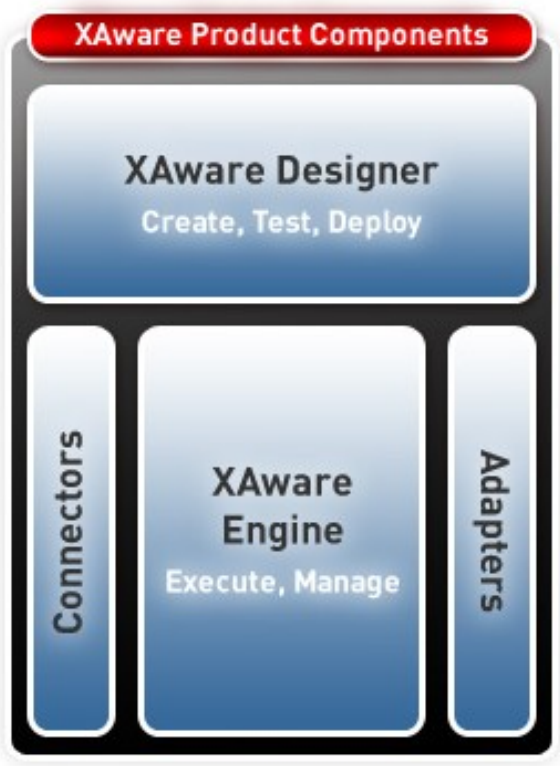


Figura 65: Componenti di XAware

può essere utilizzato direttamente da un altro programma Java;
altrimenti può essere affiancato ad un web server ed invocato tramite SOAP o semplici POST / GET HTTP;
infine può essere fatto girare all'interno di application server J2EE commerciali come WebSphere di IBM, JBOSS e altri.

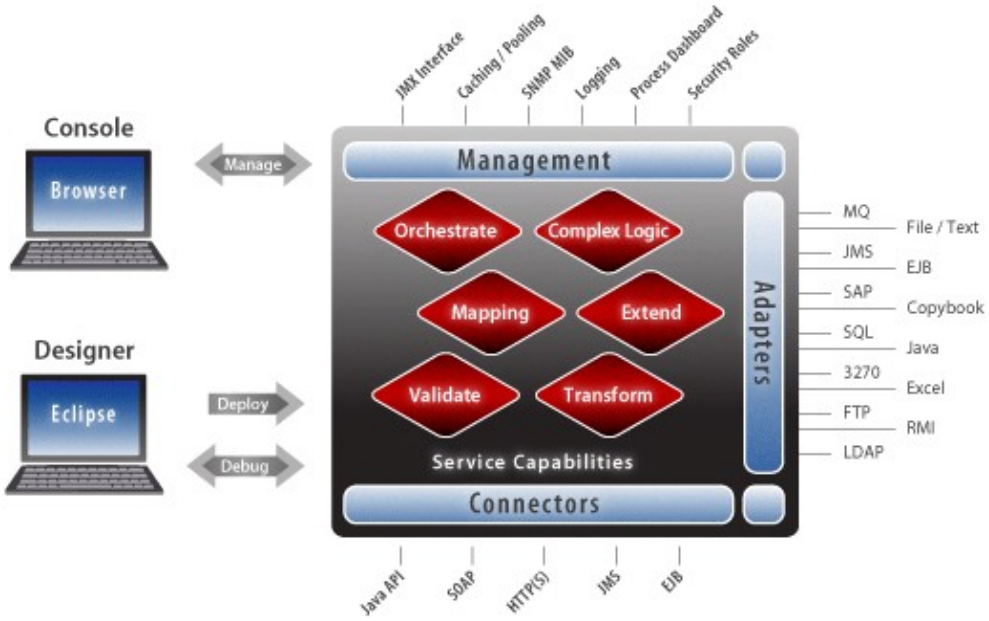


Figura 66: Interazione del XA-Engine con il mondo esterno

Come si può vedere nella figura il cuore di XA-Engine è rappresentata dall'interpreter che si occupa di eseguire il parsing delle BizView ed espanderle, se necessario, prima della loro esecuzione. L'interpreter lavora in coppia con il Resource Manager che, come ci si potrebbe aspettare dal nome, arbitra gli accessi alle risorse, alle BizView, ai file di configurazione ed ai dati. Il Request Manager ha il compito di gestire le richieste di esecuzione delle BizView in maniera classica mentre l'altra interfaccia, la Debug Interface, è utilizzata invece da Tracer, un debugger che permette di testare e controllare le view create, per poter seguire l'esecuzione passo passo.
Lo stack grigio è il blocco logico che si occupa di gestire il life cycle di una richiesta nella sua interezza ed è composto da:
il Transaction Manager che coordina le transazioni sulle varie sorgenti gestite da una BizView per mantenere consistente tutte le operazioni;
il Security Manager che esegue i controlli sui permessi di esecuzione sui BizComponent interfacciandosi con le ACL o LDAP;
il Channel Manager che amministra i canali di comunicazione e i vari

protocolli con cui ci si collega alle sorgenti, come HTTP o connessioni ai database, accesso al disco ecc;
il modulo di Initialization che si occupa di inizializzare il sistema decidendo quali file di configurazione e in che ordine leggerli;
Aliasing che traduce i nomi e i riferimenti in uno fully qualified e viceversa, permettendo di usare alias per i percorsi;
Logging che gestisce i log del sistema sfruttando Java Logging.

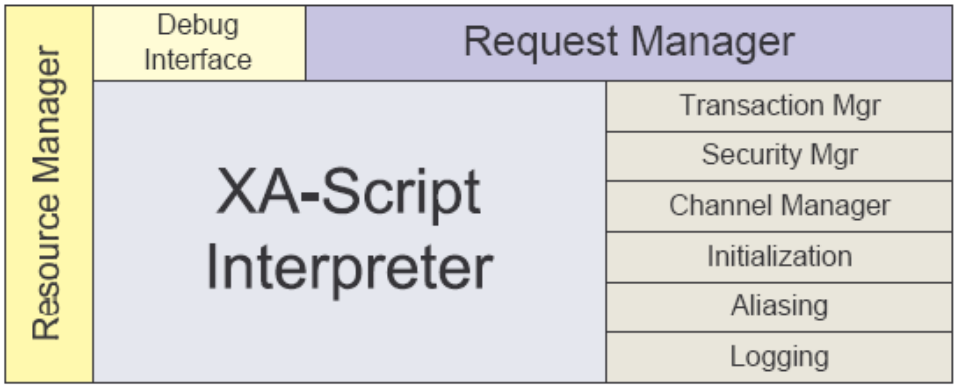


Figura 67: Architettura interna di XA-Engine

Per fare ulteriore chiarezza ricordo nuovamente che un BizDriver è un file XML che contiene le informazioni per la connessione ad una sorgente, come ad esempio la sua locazione, utente e password e così via, e non va confuso con gli Adapter.

Operativamente, una volta completato il design di una BizView con il XA-Designer tutti i vari file XML generati vengono inseriti in un archivio, chiamato XAR di cui viene eseguito poi il deployment sul XA-Engine che lo manderà poi in esecuzione. Durante la fase di archiviazione vengono stabiliti anche i permessi di accesso ed esecuzioni dei vari BizComponent.

L'esecuzione di una BizView è suddivisa i questo modo:

1. Per prima cosa viene clonato il codice della BizView
2. I file XML sono parsati alla ricerca dei tag che identificano gli Xaware

script

3. Vengono eseguiti gli script
4. Vengono riportati i risultati

8.2.3 XA-Designer

XA-Designer è un programma Java basato su Eclipse di IBM, utilizzabile per la costruzione visuale e semi-automatica delle BizView.
Essendo scritto completamente in Java non avrebbe bisogno di un'installazione vera e propria ma viene comunque distribuito come un eseguibile auto-installante che copia i vari file nel posto giusto e crea i link necessari.
L'installazione è automatica e non necessita di particolari approfondimenti, il wizard guida l'utente attraverso tutti i passi necessari; l'unica cosa da decidere è la cartella di destinazione (/opt/xaware di default per Linux) in cui si vuole installare e la cartella dove posizionare i link.

L'interfaccia ha la struttura tipica dei programmi basati su Eclipse e risulta molto simile anche a Talend (si veda sopra).

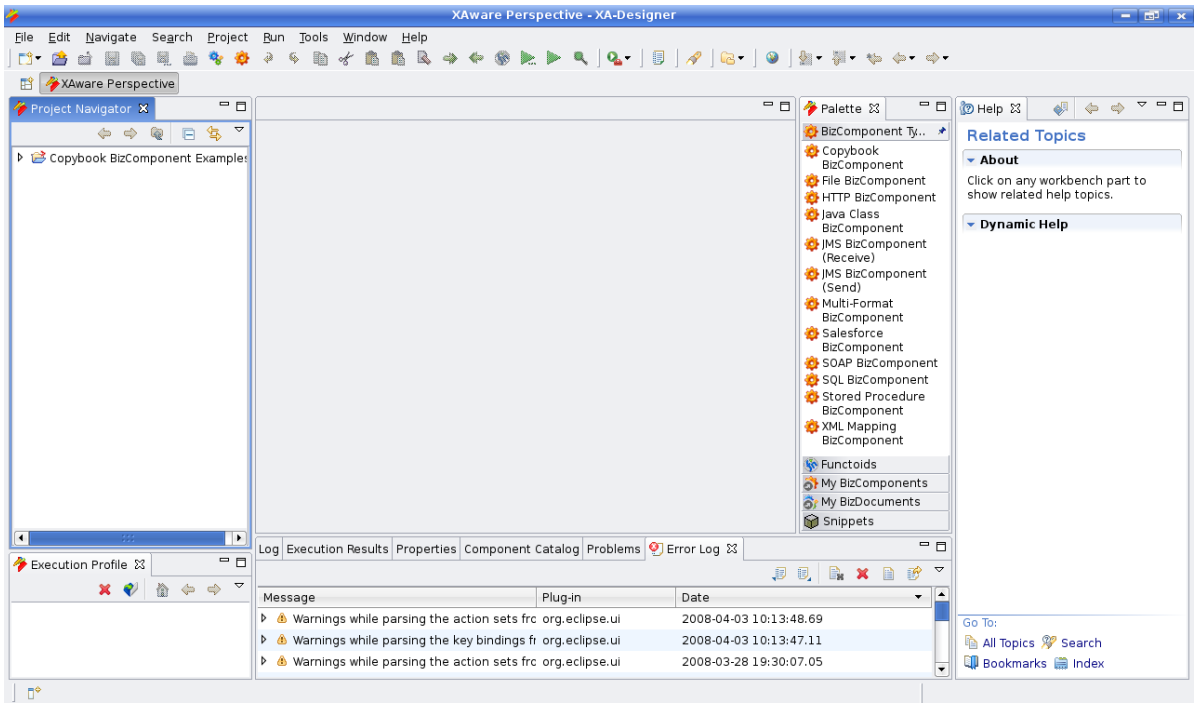


Figura 68: L'interfaccia di XA-Designer

Oltre ai soliti menu che troviamo in alto possiamo vedere nella colonna di sinistra il project navigator dove vengono elencati i progetti e i relativi file; subito sotto troviamo il pannello Execution Profile dove vengono riportati i messaggi di stato, i log e altre informazioni riguardanti l'esecuzione di una BizView. Al centro troviamo l'area principale di lavoro dove vengono visualizzati i vari file XML in lavorazione. Sotto viene mostrata una barra informativa sul file aperto nell'area centrale e subito a sinistra la palette degli strumenti utilizzata per costruire velocemente i file XML necessari.
All'estrema destra la colonna con l'help, non visualizzata di default.

La palette è divisa in drawer, cassetti, contenenti ciascuno strumenti specifici a seconda dell'attività. La palette è molto semplice da usare e la maggior parte dei tool è auto esplicativo a parte due che possono apparire poco chiari: gli snippets e i functoids.
Gli snippets sono banalmente dei template già pronti per i casi più comuni, i Functoids invece non sono altro che delle funzioni di trasformazione degli

attributi, utili per manipolare attraverso Java o Javascript i campi XML.

Nella figura sottostante viene mostrato come appare un nuovo BizDocument all'interno del XA-Designer nella design view:

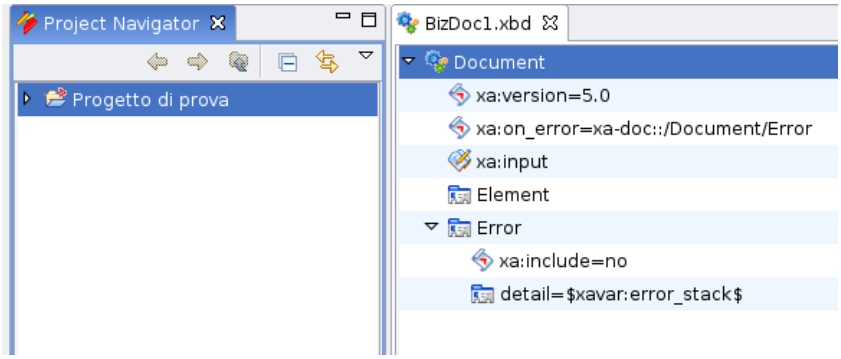


Figura 69: Design view di un BizDocument dentro a XA-Designer

E lo stesso BizDocument nella XML view:

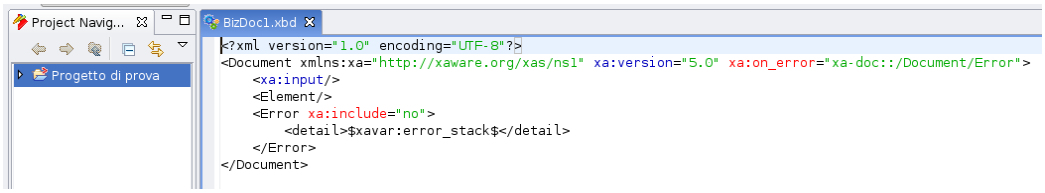


Figura 70: L'XML dello stesso BizDocument

L'editing del BizDocument può essere portato avanti o in maniera visuale nella modalità design view oppure manualmente andando ad agire direttamente sul XML.

Utilizzando la design view si possono manipolare e aggiungere gli attributi necessari attraverso dei semplici wizard; per aggiungere un attributo ad esempio è sufficiente cliccare con li tasto destro sulla radice del documento e

selezionare Add attribute; a questo punto è possibile inserire un attributo attraverso un menu a tendina e i valori consentiti per quel attributo.

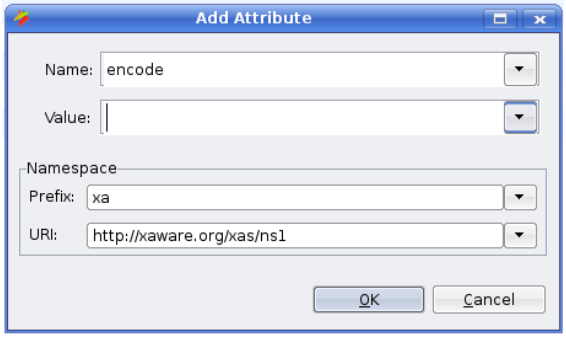


Illustration 2: Aggiunta di un attributo

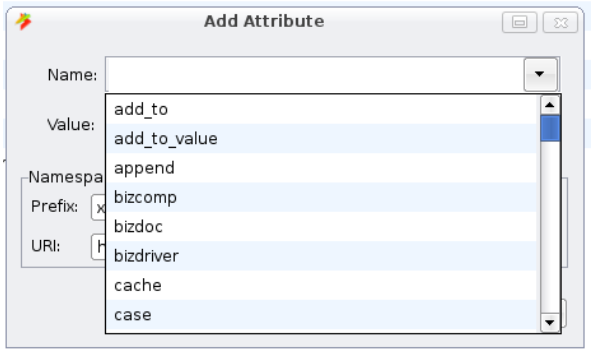


Illustration 1: Selezione dell'attributo

9 Conclusioni

Benché giovane il mercato della Data Integration risulta già particolarmente agguerrito, soprattutto se si considerano prodotti proprietari, con i principali vendor già entrati nel mercato e che puntano alla conquista di fette di mercato sempre maggiori.

Al momento tuttavia le offerte commerciali offrono soluzioni concrete ma non particolarmente avanzate dal punto di vista tecnologico, limitandosi per lo più a procedure manuali o guidate e semi-automatiche; i margini di miglioramento risultano ampi e in futuro è facile aspettarsi che i sistemi della Data Integration si evolveranno verso soluzioni più intelligenti e autonome, permettendo di integrare un crescente numero di sorgenti dati rapidamente e con minor sforzo.

I software Open Source per la Data Integration vagliati in questo elaborato non sono al momento a livello dei più diffusi prodotti commerciali, come quelli di IBM o di Informatica, per volume di vendite e per fatturato, anche se è possibile aspettarsi che ne diventino, già in un paio di anni, i principali concorrenti. Come si può notare inoltre il mondo Open Source risulta in questo momento particolarmente attivo e in fermento, consentendo ancora ampi margini di sviluppo ed esprimendo una forte capacità di attrarre utenti e sviluppatori.

Il contributo principale di questa tesi è stato quello di poter individuare una strategia per portare il sistema MOMIS nel mondo del software commerciale, in particolare puntando inizialmente alla business intelligence, e le aree da potenziare per potere in futuro offrire una soluzione di livello enterprise.

Come visto MOMIS potrebbe trarre ampi benefici dalla scelta di rilasciare con licenza Open Source tutto il codice sviluppato, consentendo di accelerare lo sviluppo e migliorando contemporaneamente la qualità del codice. Nello specifico la scelta che sarebbe più efficace ricade sulla GPL versione 2 che garantisce l'accesso al codice sorgente da parte di terzi ma, contemporaneamente, assicura che nessuno possa avvantaggiarsi del lavoro

svolto senza dare nulla in cambio.

Le altri licenze Open Source, come la Apache License o la BSD, non proteggono dal rischio di Free Riding e non è pertanto consigliato utilizzarle per la commercializzazione di questo prodotto; nel caso, al contrario, si voglia pubblicare DataRiver per puri scopi accademici, una licenza simil-BSD, scelta adottata anche da WordNet, risulterebbe la più efficace.

Per raggiungere questo risultato sono state analizzate le funzionalità di DataRiver, per individuare i punti di forza su cui puntare per una eventuale commercializzazione, e l'architettura, per poter decidere se e quali parti del programma rendere effettivamente Open Source.

E' stata inoltre effettuata una concisa analisi delle offerte attualmente disponibili sul mercato e delineando quali potrebbe essere gli sviluppi futuri.

Un aspetto su cui si potrebbe puntare, e che sembra l'area di maggiore espansione per l'immediato futuro, è quella attualmente occupata da soluzioni sviluppate internamente da parte dei sistemisti delle aziende; al momento è infatti stimato che riuscendo a penetrare in questo segmento si potrebbe sbloccare una fetta di mercato di valore pari al valore dell'intero mercato della Data Integration attuale.

E' prudente in ogni caso aspettarsi che la richiesta di soluzioni per la Data Integration continui ad aumentare nei prossimi anni: da un lato la richiesta sarà guidata dal naturale aumento del volume di dati dall'altro da una maggiore consapevolezza dei clienti.

Se infatti il volume dei dati è destinato ad aumentare ancora a lungo, sia perché aumentano le attività che vengono smaterializzate e informatizzate sia perché aumentano gli utenti di questi servizi, è inoltre evidente che le aziende con il tempo si trovano a gestire infrastrutture informatiche sempre più complesse.

Gli sviluppi futuri per DataRiver potrebbero essere molteplici a seconda del tipo di offerta che si vuole proporre; nel caso si decida di affrontare il

mercato enterprise, come visto in questo elaborato, sarà necessario potenziare tutti i servizi accessori che permettano di integrare il programma negli scenari più comuni in ambiente enterprise, fornendo soprattuto migliori servizi di autenticazione, autorizzazione e visibilità dei dati.

In ogni caso DataRiver potrebbe riuscire a guadagnare un certo seguito ed è di primaria importanza imprimere un impulso per accelerare lo sviluppo e raggiungere il mercato in tempi brevi.

10 Appendice A

Qui di seguito viene riportato il confronto la versione 2 la versione della GPL; purtroppo non esistono traduzioni ufficiali in italiano delle licenze e bisogna pertanto rifarsi a quelle in lingua inglese. L'impaginazione e disposizione delle licenze è stata fatta in maniera tale da poter eseguire un confronto puntuale clausola per clausola ed evidenziando i passaggi che sono stati modificati.

#	GPLv2	GPLv3
-	**Preamble**	**Preamble**
-	The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.	The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other program whose authors commit to using it. (Some Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.
-	When we speak of free software, we are referring to freedom, not	When we speak of free software, we are referring to freedom, not price.

	price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.	Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
-	To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.	To protect your rights, we need to make requirements that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
-	For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.	For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
-	We protect your rights with two steps: (1) copyright the software,	Developers that use the GNU GPL protect your rights with two steps: (1)

	and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.	assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.
-	Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.	For the developers' and author's protection, the GPL clearly explains that there is no warranty for this free software. If the software is modified by someone else and passed on, the GPL ensures that recipients are told that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
-	- -	Some countries have adopted laws prohibiting software that enables users to escape from Digital Restrictions Management. DRM is fundamentally incompatible with the purpose of the GPL, which is to protect users' freedom; therefore, the GPL ensures that the software it covers will neither be subject to, nor subject other works to, digital restrictions from which escape is forbidden.
-	Finally, any free program is threatened constantly by	Finally, every program is threatened constantly by software patents. We

	software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.	wish to avoid the special danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, the GPL makes it clear that any patent must be licensed for everyone's free use or not licensed at all.
-	The precise terms and conditions for copying, distribution and modification follow.	The precise terms and conditions for copying, distribution and modification follow.
-	**GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**	**GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**
0	This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or	Definitions. A "licensed program" means any program or other work distributed under this License.The "Program" refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either modified or unmodified. Throughout this License, the term "modification" includes, without limitation, translation and extension. A "covered

	translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you". Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.	work" means either the Program or any work based on the Program. Each licensee is addressed as "you". To "propagate" a work means doing anything with it that requires permission under applicable copyright law, other than executing it on a computer or making private modifications. This includes copying, distribution (with or without modification), sublicensing, and in some countries other activities as well.
1	- -	Source Code. The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source version of a work. The "Complete Corresponding Source Code" for a work in object code form means all the source code needed to understand, adapt, modify, compile, link, install, and run the work, excluding general-purpose tools used in performing those activities but

		which are not part of the work. For example, this includes any scripts used to control those activities, and any shared libraries and dynamically linked subprograms that the work is designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work, and interface definition files associated with the program source files. Complete Corresponding Source Code also includes any encryption or authorization codes necessary to install and/or execute the source code of the work, perhaps modified by you, in the recommended or principal context of use, such that its functioning in all circumstances is identical to that of the work, except as altered by your modifications. It also includes any decryption codes necessary to access or unseal the work's output. Notwithstanding this, a code need not be included in cases where use of the work normally implies the user already has it. Complete Corresponding Source Code need not include anything that users

		can regenerate automatically from other parts of the Complete Corresponding Source Code. As a special exception, the Complete Corresponding Source Code need not include a particular subunit if (a) the identical subunit is normally included as an adjunct in the distribution of either a major essential component (kernel, window system, and so on) of the operating system on which the executable runs or a compiler used to produce the executable or an object code interpreter used to run it, and (b) the subunit (aside from possible incidental extensions) serves only to enable use of the work with that system component or compiler or interpreter, or to implement a widely used or standard interface, the implementation of which requires no patent license not already generally available for software under this License.
2	- -	Basic Permissions. All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms

		your unlimited permission to run the Program. The output from running it is covered by this License only if the output, given its content, constitutes a work based on the Program. This License acknowledges your rights of "fair use" or other equivalent, as provided by copyright law. This License gives unlimited permission to privately modify and run the Program, provided you do not bring suit for patent infringement against anyone for making, using or distributing their own works based on the Program. Propagation of covered works is permitted without limitation provided it does not enable parties other than you to make or receive copies. Propagation which does enable them to do so is permitted, as "distribution", under the conditions of sections 4-6 below.
3	- -	Digital Restrictions Management. As a free software license, this License intrinsically disfavors technical attempts to restrict users' freedom to copy, modify, and share copyrighted works. Each of its provisions shall be

		interpreted in light of this specific declaration of the licensor's intent. Regardless of any other provision of this License, no permission is given to distribute covered works that illegally invade users' privacy, nor for modes of distribution that deny users that run covered works the full exercise of the legal rights granted by this License. No covered work constitutes part of an effective technological protection measure: that is to say, distribution of a covered work as part of a system to generate or access certain data constitutes general permission at least for development, distribution and use, under this License, of other software capable of accessing the same data.
4	1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty ; keep intact all the notices that refer to this License and to the absence	Verbatim Copying. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all license notices and notices of the absence of any warranty; give all recipients of the Program a copy of

	of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.	this License along with the Program; and obey any additional terms present on parts of the Program in accord with section 7. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection for a fee.
5	2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions: a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this	Distributing Modified Source Versions. Having modified a copy of the Program under the conditions of section 2, thus forming a work based on the Program, you may copy and distribute such modifications or work in the form of source code under the terms of Section 4 above, provided that you also meet all of these conditions: a) The modified work must carry prominent notices stating that you changed the work and the date of any change. b) You must license the entire modified work, as a whole, under this License to anyone who comes into possession of a copy. This License

License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms,	must apply, unmodified except as permitted by section 7 below, to the whole of the work. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. c) If the modified work has interactive user interfaces, each must include a convenient feature that displays an appropriate copyright notice, and tells the user that there is no warranty for the program (or that you provide a warranty), that users may redistribute the modified work under these conditions, and how to view a copy of this License together with the central list (if any) of other terms in accord with section 7. If the interface presents a list of user commands or options, such as a menu, a command to display this information must be prominent in the list. Otherwise, the modified work must display this information at startup--except in the case that the Program has such interactive modes and does not display this information at startup. These requirements apply to the modified work as a whole. If

	do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.	identifiable sections of that work, added by you, are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works for use not in combination with the Program. But when you distribute the same sections for use in combination with covered works, no matter in what form such combination occurs, the whole of the combination must be licensed under this License, whose permissions for other licensees extend to the entire whole, and thus to every part of the whole. Your sections may carry other terms as part of this combination in limited ways, described in section 7. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. A compilation of a covered work with other separate and independent works, which are not by their nature

		extensions of the covered work, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. Mere inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.
6	3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following: a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, b) Accompany it with a written offer, valid for at least three years, to give any third party, for	Non-Source Distribution. You may copy and distribute a covered work in Object Code form under the terms of Sections 4 and 5, provided that you also distribute the machine-readable Complete Corresponding Source Code (herein the "Corresponding Source") under the terms of this License, in one of these ways: a) Distribute the Object Code in a physical product (including a physical distribution medium), accompanied by the Corresponding Source distributed on a durable physical medium customarily used for software interchange; or,

a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a	b) Distribute the Object Code in a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give any third party, for a price no more than ten times your cost of physically performing source distribution, a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange; or, c) Privately distribute the Object Code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only for occasional noncommercial distribution, and only if you received the Object Code with such an offer, in accord with Subsection b above. Or, d) Distribute the Object Code by offering access to copy it from a designated place, and offer equivalent access to copy the Corresponding Source in the same way through the same place. You need not require

	special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.	recipients to copy the Corresponding Source along with the Object Code. [If the place to copy the Object Code is a network server, the Corresponding Source may be on a different server that supports equivalent copying facilities, provided you have explicitly arranged with the operator of that server to keep the Corresponding Source available for as long as needed to satisfy these requirements, and provided you maintain clear directions next to the Object Code saying where to find the Corresponding Source.] Distribution of the Corresponding Source in accord with this section must be in a format that is publicly documented, unencumbered by patents, and must require no special password or key for unpacking, reading or copying. The Corresponding Source may include portions which do not formally state this License as their license, but qualify under section 7 for inclusion in a work under this License.
7	- -	License Compatibility.

When you release a work based on the Program, you may include your own terms covering added parts for which you have, or can give, appropriate copyright permission, as long as those terms clearly permit all the activities that this License permits, or permit usage or relicensing under this License. Your terms may be written separately or may be this License plus additional written permission. If you so license your own added parts, those parts may be used separately under your terms, but the entire work remains under this License. Those who copy the work, or works based on it, must preserve your terms just as they must preserve this License, as long as any substantial portion of the parts they apply to are present.

Aside from additional permissions, your terms may add limited kinds of additional requirements on your added parts, as follows:

a) They may require the preservation of certain copyright notices, other legal notices, and/or author attributions, and may require that the origin of the parts they cover not be misrepresented, and/or that altered

versions of them be marked in the source code, or marked there in specific reasonable ways, as different from the original version.

b) They may state a disclaimer of warranty and liability in terms different from those used in this License.

c) They may prohibit or limit the use for publicity purposes of specified names of contributors, and they may require that certain specified trademarks be used for publicity purposes only in the ways that are fair use under trademark law except with express permission.

d) They may require that the work contain functioning facilities that allow users to immediately obtain copies of its Complete Corresponding Source Code.

e) They may impose software patent retaliation, which means permission for use of your added parts terminates or may be terminated, wholly or partially, under stated conditions, for users closely related to any party that has filed a software patent lawsuit (i.e., a lawsuit alleging

		that some software infringes a patent). The conditions must limit retaliation to a subset of these two cases: 1. Lawsuits that lack the justification of retaliating against other software patent lawsuits that lack such justification. 2. Lawsuits that target part of this work, or other code that was elsewhere released together with the parts you added, the whole being under the terms used here for those parts. No other additional conditions are permitted in your terms; therefore, no other conditions can be present on any work that uses this License. This License does not attempt to enforce your terms, or assert that they are valid or enforceable by you; it simply does not prohibit you from employing them. When others modify the work, if they modify your parts of it, they may release such parts of their versions under this License without additional permissions, by including notice to that effect, or by deleting the notice that gives specific permissions in addition to this License. Then any broader permissions granted by your

		terms which are not granted by this License will not apply to their modifications, or to the modified versions of your parts resulting from their modifications. However, the specific requirements of your terms will still apply to whatever was derived from your added parts. Unless the work also permits distribution under a previous version of this License, all the other terms included in the work under this section must be listed, together, in a central list in the work.
8	4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.	Termination. You may not propagate, modify or sublicense the Program except as expressly provided under this License. Any attempt otherwise to propagate, modify or sublicense the Program is void, and any copyright holder may terminate your rights under this License at any time after having notified you of the violation by any reasonable means within 60 days of any occurrence. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as

		they remain in full compliance.
9	5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.	Not a Contract. You are not required to accept this License in order to receive a copy of the Program. However, nothing else grants you permission to propagate or modify the Program or any covered works. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating the Program (or any covered work), you indicate your acceptance of this License to do so, and all its terms and conditions.
10	6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to	Automatic Licensing of Downstream Users. Each time you redistribute a covered work, the recipient automatically receives a license from the original licensors, to propagate and modify that work, subject to this License, including any additional terms introduced through section 7. You may not impose any further restrictions on the recipients' exercise of the rights thus granted or affirmed,

	this License.	except (when modifying the work) in the limited ways permitted by section 7. You are not responsible for enforcing compliance by third parties to this License.
11	--	Licensing of Patents. When you distribute a covered work, you grant a patent license to the recipient, and to anyone that receives any version of the work, permitting, for any and all versions of the covered work, all activities allowed or contemplated by this License, such as installing, running and distributing versions of the work, and using their output. This patent license is nonexclusive, royalty-free and worldwide, and covers all patent claims you control or have the right to sublicense, at the time you distribute the covered work or in the future, that would be infringed or violated by the covered work or any reasonably contemplated use of the covered work. If you distribute a covered work knowingly relying on a patent license, you must act to shield downstream users against the possible patent infringement claims from which your

		license protects you.
1 2	7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program. If any portion of this section is held invalid or unenforceable under any particular	Liberty or Death for the Program. If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute the Program, or other covered work,so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute it at all. For example, if a patent license would not permit royalty-free redistribution by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution. It is not the purpose of this section to induce you to infringe any patents or other exclusive rights or to contest their legal validity. The sole purpose of this section is to protect the integrity of the free software distribution system. Many people have made generous contributions to the wide range of software distributed

	circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances. It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.	through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

13	8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.	[Geographical Limitations. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.]
14	9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and	Revised Versions of this License. The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of this License "or any later version" applies to it, you have the option of following the terms and

	conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.	conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
15	10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.	Requesting Exceptions. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.
16	NO WARRANTY 11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE	NO WARRANTY There is no warranty for the Program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the Program "as

	LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.	is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.
17	12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR	In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the Program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the Program (including but not limited to loss of data or data being

	CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.	rendered inaccurate or losses sustained by you or third parties or a failure of the Program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.
18	- -	Unless specifically stated, the Program has not been tested for use in safety critical systems.

11 Appendice B

WordNet Release 3.0

This software and database is being provided to you, the LICENSEE, by Princeton University under the following license. By obtaining, using and/or copying this software and database, you agree that you have read, understood, and will comply with these terms and conditions.:

Permission to use, copy, modify and distribute this software and database and its documentation for any purpose and without fee or royalty is hereby granted, provided that you agree to comply with the following copyright notice and statements, including the disclaimer, and that the same appear on ALL copies of the software, database and documentation, including modifications that you make for internal use or for distribution.

WordNet 3.0 Copyright 2006 by Princeton University. All rights reserved.

THIS SOFTWARE AND DATABASE IS PROVIDED "AS IS" AND PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANT-
ABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

The name of Princeton University or Princeton may not be used in advertising or publicity pertaining to distribution of the software and/or database. Title to copyright in this software, database and

any associated documentation shall at all times remain with Princeton University and LICENSEE agrees to preserve same.

12 Indice delle figure

13 Bibliografia

[a] http://www.xaware.org

[b] http://www.xaware.com

[c] http://www.hoovers.com

[d] http://opensource.sys-con.com

[e] http://opensource.org

[f] https://www.linux-foundation.org/

[g] http://wikiquote.org

[h] http://gpl3.palamida.com

[i] http://dns.measurement-factory.com

[l] http://counter.li.org/

[m] http://www.informatica.com

[n] http://devnet.informatica.com

[o] http://www.s1.com/

[p] http://www.linuxinsider.com

[q] http://soa.sys-con.com

[r] http://www.businesswire.com

[t] http://investing.businessweek.com

[u] http://www.computerworld.com.au

[v] http://www.talend.com/

[1] Classificazione del Software Libero e non libero Free Software Foundation

[2] Permesso d'autore: idealismo pragmatico di Richard Stallman

[3] Voci dalla rivoluzione Open Source autori vari

[4] La saggezza della folla - di James Surowiecki

[5] The Emerging Economic Paradigm of Open Source di Bruce Perens

[6] Free/Libre and Open Source Software: Survey and Study Università di Maastricht

[7] Designing brand identity di Alina Wheeler
[8] Analisi e valutazione comparativa dei principali sistemi di integrazione dati commerciali rispetto al sistema Momis attraverso il benchmark Thalia di Fabio Romano