# UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA QUINQUENNALE IN INFORMATICA


## VIRTUOSE 3.0:
## LA NUOVA VERSIONE DI VIRTUOSE BASATA SU XML

Relatore: Prof.ssa Fiorella DE CINDIO
Correlatore: Dott. Leonardo SONNANTE

Tesi di Laurea di:
Davide AIRAGHI
Matr.n. 606120

Anno Accademico 2005-2006

*Ringrazio la Professoressa Fiorella De Cindio, Leonardo Sonnante e Michele Chinosi per la disponibilità dimostrata e per avermi permesso di lavorare su un progetto interessante quale è stato Virtuose.*

*Un "grazie" speciale va anche a mia madre, mio padre, mia sorella e mia nonna che mi hanno aiutato a tener duro nei momenti bui e a proseguire nel mio cammino anche quando le difficoltà parevano insormontabili.*

*Non possono mancare dalla lista delle persone da ringraziare i miei amici Stefano e Cristian dato che mi sono stati di supporto e compagnia in questi ultimi quattro anni di studio e lavoro.*

*E per concludere un ringraziamento collettivo anche a tutti gli appartenenti alla Repubblica Digitale di Lithium, la mia casa nella grande rete, che hanno sempre fornito ottimi spunti di riflessione e di confronto, ma anche di sano relax.*

# Indice generale

1.Introduzione..... 3
1.1. Concetti principali di Virtuose..... 5
2.Nuove funzionalità e requisiti di Virtuose 3.0..... 7
2.1 Struttura e gestione del database XML..... 7
2.2 Logica applicativa e relativa organizzazione.....10
3.Architettura di Virtuose 3.0.....13
3.1. Struttura base..... 14
3.1.1. Nucleo applicativo.....17
3.1.2. Sistema di comunicazione.....19
3.1.3. Moduli.....21
3.1.4. Gestione dei contenuti.....24
Conferenze.....25
Messaggi.....28
3.2. Principali Moduli e relative funzionalità.....31
3.2.1. Messages..... 32
3.2.2. Conferences.....35
3.2.3. Users..... 38
2.2.4 Profiles.....41
3.2.5 Views.....42
3.2.6 Storage.....44
3.3 Visualizzazione Conferenza .....47
3.4 Visualizzazione Messaggio.....49
4.Implementazione..... 51
4.1. Tecnologie scelte..... 52
4.2. Scelta Database XML..... 55
Ozone.....56
eXist .....57
Berkeley DB XML.....59
4.2.1. Struttura XML delle principali entità del database..... 61
Conference.....61
Message.....62
Profile.....63
User..... 64
View.....65
4.3. Scambio dati in XML.....68
4.4. Logica applicativa in Java.....70
it.virtuose..... 70
it.virtuose.communication..... 71
it.virtuose.modules.....73
it.virtuose.modules.storage_plugins.....74
it.virtuose.modules.user_plugins..... 75
it.virtuose.utils.....75
it.virtuose.utils.modules..... 76
it.virtuose.utils.xml.....77
it.virtuose.webcommunity..... 77

it.virtuose.webcommunity.layouts..... 78
it.virtuose.webcommunity.portlets..... 78
4.5. Realizzazione layout tramite XSL..... 81
4.6. Uso di Jetspeed..... 85
4.6.1. Integrazione con Jetspeed..... 85
4.6.2. Portlet..... 89
4.6.3. Sistema di comunicazione tra Portlet..... 93
4.6.4. Template grafici di Jetspeed..... 96
5.Conclusioni..... 100
5.1. Sviluppi futuri..... 103
Bibliografia..... 106

# 1. Introduzione

Questa tesi vuole portare avanti il lavoro svolto prima da Simone Fedele [FED] e poi da Michele Chinosi [CHI04] volto a trasformare il database del software di gestione di comunità online Virtuose [BDS05] [VIRT] in un sistema basato completamente su XML [YBP04].

L'obiettivo finale risulta quindi essere la realizzazione di un sistema interamente incentrato intorno a questa tecnologia sia per quanto riguarda la gestione e memorizzazione dei dati sia per lo scambio e l'elaborazione di informazioni tra i diversi moduli sia per la creazione e visualizzazione dell'interfaccia utente finale.

Il lavoro è stato svolto in varie fasi:

- studio dei requisiti di base da soddisfare e delle funzionalità necessarie
- pianificazione dell'architettura generale su cui basare l'applicazione
- scelta delle tecnologie da utilizzare per l'implementazione
- implementazione (nel precedente lavoro di tesi di Michele Chinosi si era giunti ad avere solo una proposta di organizzazione dei dati via XML e un insieme basilari di strumenti per effettuare interrogazioni su di essi).

Per la fase iniziale si è partiti dall'analisi dei difetti e delle carenze di funzionalità presenti nelle versioni 1.0 e 2.0 di Virtuose in modo da poterli superare e poi si è passati ad introdurre tutto quanto fosse utile per avere una nuova applicazione caratterizzata da maggiore flessibilità e capacità di espansione, oltre ad esse facilmente integrabile con altre soluzioni applicative già presenti.

Per la pianificazione dell'architettura ci si è mossi attorno al concetto di modularità in quanto tramite un'architettura modulare è possibile realizzare un'applicazione in grado di

poter supportare lo sviluppo di nuove funzionalità.

Partendo dalla suddivisione del sistema in moduli è stata ideata una struttura orientata alla

suddivisione in unità di elaborazione, altamente specializzate nell'implementazione delle differenti funzionalità, in grado di nascondere completamente la propria logica applicativa e dotate di una semplice interfaccia tramite la quale permettere la comunicazione. La scelta delle tecnologie da utilizzare, alla fine delle valutazioni, ha visto prevalere per la scrittura del codice applicativo il linguaggio Java [ECK06], per l'organizzazione dei dati e delle comunicazioni tra componenti del sistema XML e per la creazione dell'interfaccia utente XSLT [VOT02].

L'uso dell'XML oltre a rendere uniforme la gestione delle informazioni nel database e nello scambio di informazioni tra moduli ha permesso di aggiungere indicazioni semantiche e organizzative alla struttura dei dati, attraverso l'uso di tag e attributi con opportuni nomi e

struttura, rendendone così più semplice la gestione e l'elaborazione.

Per la parte di implementazione si è poi deciso di far affidamento, oltre che sulle tecnologie

scelte, anche su un framework espressamente orientato alla realizzazione di applicazioni web quale è Jetspeed 2 [AJ2].

Jetspeed si è rivelato uno strumento altamente personalizzabile e flessibile per l'organizzazione della struttura delle varie pagine html necessarie per l'interazione degli utenti con il sistema e ci ha consentito di focalizzare questo lavoro sulla progettazione e implementazione del cuore del sistema come verrà descritto nei capitoli successivi.

Prima di proseguire è necessario dare qualche informazione ulteriore su Virtuose, almeno per quanto riguarda i concetti chiave che lo hanno sempre caratterizzato e che saranno anche parte dei pilastri della nuova versione.

### *1.1. Concetti principali di Virtuose*

Alla base di Virtuose, strumento inizialmente orientato in modo specifico alla gestione di una Comunità Virtuale [RHE04], è presente il concetto di Messaggio, inteso come contributo di un determinato tipo inserito da un Utente, contenuto in una Conferenza, entità avente il ruolo di vero e proprio luogo di memorizzazione e organizzazione dei contenuti.

Il concetto di Tipo di Messaggio gioca un ruolo fondamentale all'interno di Virtuose in quanto attraverso di esso possono essere definite le caratteristiche a livello strutturale e di semantica che un messaggio associato ad esso presenta e quindi si ha modo di ottimizzare l'organizzazione dei contenuti presenti nel sistema.

La definizione di una serie di regole atte a regolamentare le possibilità d'azione di un utente su messaggi e conferenze è delegata al concetto di Vista; questa non è direttamente in relazione con l'utente ma lo diventa attraverso un passo intermedio, l'uso di un Profilo definito come elemento per raggruppare più utenti all'interno di uno stesso gruppo caratterizzato dagli stessi permessi e possibilità d'azione.

All'interno della vista troviamo codificate le azioni consentite per un dato profilo sui messaggi, oltre che sui loro contenitori, presenti all'interno delle conferenze, il tutto raggruppato per conferenza e per tipo di messaggio. Attraverso quanto indicato da tali permessi si è in grado di stabilire se un utente può leggere un messaggio, scrivere un nuovo messaggio, modificare un proprio messaggio, eliminare un proprio messaggio, eliminare un messaggio indipendentemente dall'autore ed infine modificare un messaggio indipendentemente dall'autore dello stesso.

Le possibilità d'azione di un utente sui contenuti sono state racchiuse nelle viste e non direttamente in conferenze e messaggi per motivi di suddivisione dei compiti e di semplificazione e specializzazione delle varie entità; in questo modo la gestione dei

permessi risulta più semplice e meno problematica in quanto necessita di interventi limitati anche nei casi in cui siano tanti i soggetti coinvolti.

Tutti questi concetti possono essere visti in relazione tra loro come appare nella seguente figura.

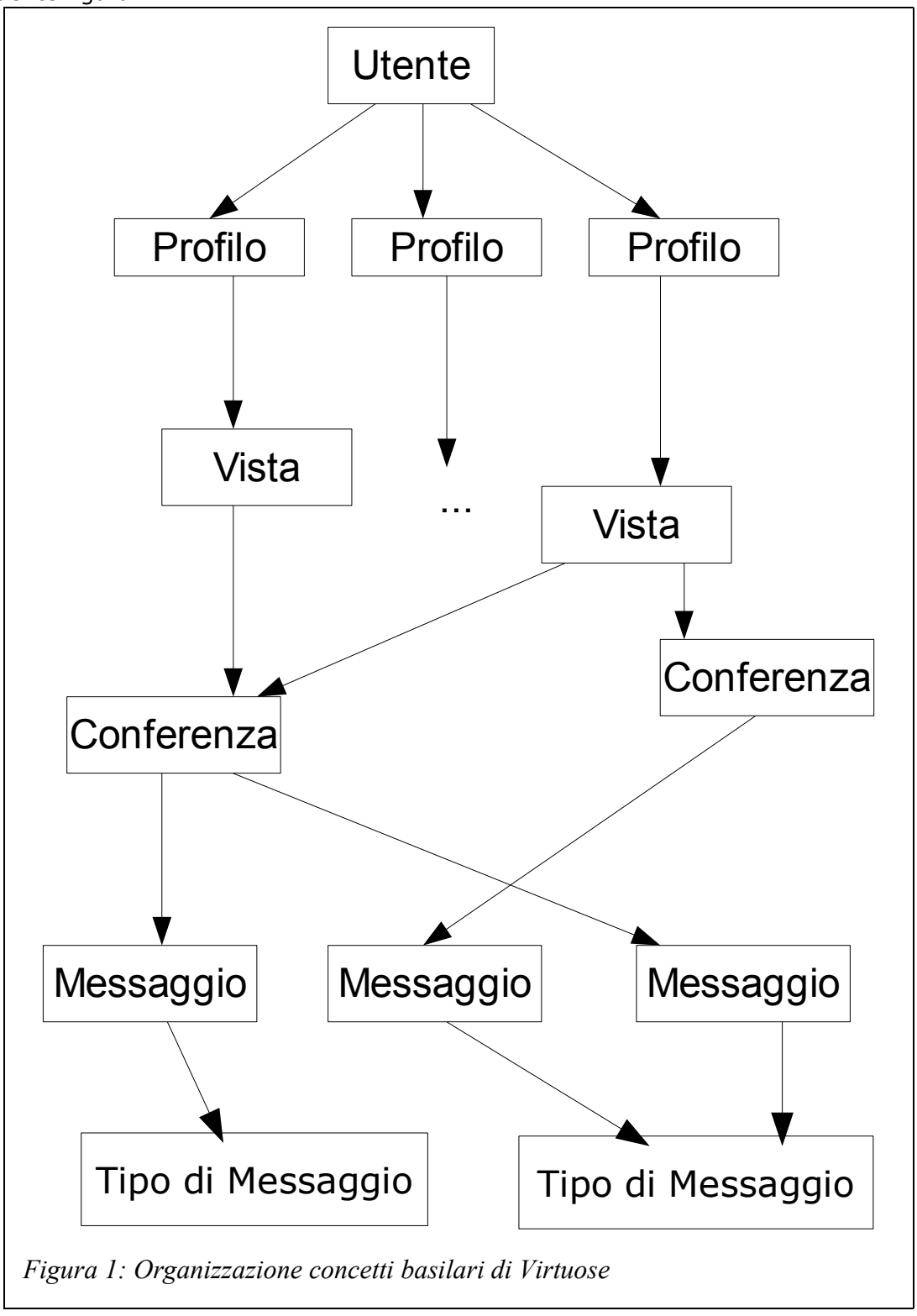


*Figura 1: Organizzazione concetti basilari di Virtuose*

# 2. Nuove funzionalità e requisiti di Virtuose 3.0

Con la versione 3.0 si è cercato di portare Virtuose ad essere un'applicazione facilmente personalizzabile, composta da moduli intercambiabili, con alte possibilità di espandibilità, in grado di essere integrata con soluzioni software già presenti e basata su tecnologie attualmente di larga diffusione e con un alto grado di affidabilità e versatilità.

La fase di pianificazione delle nuove funzionalità e dei requisiti da soddisfare nella nuova versione di Virtuose può essere vista come divisa in due ambiti distinti, uno dedicato alla base di dati e l'altro alla logica applicativa e alla sua organizzazione. Ciò è dovuto al fatto che questi due aspetti richiedono considerazioni e analisi differenti e ben distinte.

### 2.1 Struttura e gestione del database XML

In questo caso lo studio è stato orientato all'individuazione di quali requisiti e caratteristiche la struttura della base di dati ed il relativo software di gestione dovessero avere per permettere di avere poi determinate funzionalità a livello applicativo.

Il primo nodo da sciogliere è stato quello di effettuare la scelta di quale database utilizzare per la gestione dei dati XML in quanto la proposta derivante dalla precedente tesi di M. Chinosi di basarsi su Xindice [XIN], progetto della Apache Foundation, ha dovuto essere rivista causa lo sviluppo pressoché interrotto di tale software.

Seppur il fatto di scegliere direttamente in fase di studio il software possa sembrare solo un dettaglio implementativo in realtà ciò porta con se un'implicazione piuttosto importante già a livello di progettazione, infatti in questo modo si sapranno già, in base alle specifiche caratteristiche di ciascun software di gestione del database XML, quali possibili tecnologie utilizzare e quale grado di versatilità sarà presente e questo non farà altro che rendere più mirato, e sostanzialmente migliore, tutto il resto della pianificazione sulla rappresentazione dei dati.

Questo primo ostacolo ha richiesto un lavoro di ricerca per ottenere una lista di possibili candidati tra i quali selezionare un prodotto utilizzabile per i nostri scopi; questo processo ha portato all'individuazione di eXist [EXI], Berkeley DB XML [BDB] ed Ozone [OZO], tutti e tre utilizzabili attraverso opportune librerie: la scelta finale è caduta su eXist (per le motivazioni dettagliate si veda il capitolo *Implementazione*), software in continuo sviluppo attivo e con supporto per il linguaggio di interrogazione XQuery [BRU04], molto potente ed in grado di ridurre il numero di singole query nei casi in cui si debbano eseguire richieste complesse sui dati.

Arrivati a questo punto si è avuta la garanzia di poter contare su un sistema avanzato ed in grado di far fronte ad eventuali scelte strutturali del database utili dal punto di vista logico e applicativo ma realizzabili solo attraverso l'uso di costrutti ad elevato grado di complessità.

Vediamo ora quali requisiti e miglioramenti rispetto al passato si è deciso di introdurre nella struttura di conferenze, messaggi, viste e delle altre entità fondamentali alla base di Virtuose.

Per le *conferenze* si è sentita la necessità di introdurre:

- la possibilità di organizzare le stesse secondo uno schema gerarchico in modo da poter dare ordine ad esse e ai rispettivi contenuti in un ottica di sempre maggior specializzazione per quanto riguarda l'ambito dei dati in esse presenti al crescere della "profondità" nella struttura così formata
- la possibilità di creare raggruppamenti di conferenze non basati su una relazione di tipo gerarchica in modo da poter avere un ulteriore livello organizzativo trasversale
- la possibilità di avere delle conferenze il cui ruolo fosse quello di segnaposto (concetto simile ai *link simbolici* dei filesystem in stile unix), avendo così modo di riproporre una stessa conferenza in più posizioni nella struttura gerarchica senza duplicarne i dati

Per quanto riguarda i *messaggi* l'unico nuovo requisito introdotto, rispetto a quanto già presente nella versione 2.0, è il poter indicare se un messaggio faccia riferimento ad un altro: questo permetterebbe di modellare al meglio una serie di contenuti rappresentanti, ad esempio, una discussione circa un dato argomento.

Altro concetto chiave che ha risentito di questa nuova fase di analisi è quello di *vista*, il quale ha mostrato qualche limite strutturale e per questo , data la sua natura di elemento centrale in tutto il funzionamento di Virtuose, si è deciso di rivederlo completamente arrivando a formulare il suo nuovo ruolo, estensione del precedente.

Nella sua nuova implementazione la vista:

- dovrà essere in grado gestire sia i permessi relativi a messaggi e sottoconferenze, per queste ultime solo la possibilità di visualizzazione dell'anteprima, appartenenti ad una data conferenza, sia l'associazione di template grafici alle varie operazioni eseguibili di essi.
- avrà più permessi relativi ai messaggi: visione dell'anteprima messaggio (introdotto in questa nuova versione di Virtuose), lettura, scrittura, modifica proprio messaggio,eliminazione proprio messaggio, eliminazione messaggio indipendentemente dall'autore, modifica messaggio indipendentemente dall'autore dello stesso
- contenere indicazioni relative a quale modello grafico utilizzare per la visualizzazione e anteprima di conferenze, sottoconferenze e messaggi
- contenere informazioni relative alla modalità di estrazione e presentazione di messaggi associati ad una conferenza
- indicare la modalità con la quale estrarre dal database ed in seguito visualizzare eventuali messaggi riferiti a quello in esame

Un altro aspetto su cui si è lavorato e sul quale sono stati trovati nuovi requisiti da soddisfare è quello relativo ai dati degli *utenti*. In questo specifico caso si è deciso di

dare un solo nuovo requisito da soddisfare: la possibilità di utilizzare diverse fonti per i dati degli utenti. Sfruttando questa nuova possibilità sarà possibile fornire accesso al sistema a tutta una serie do soggetti non per forza gestiti in locale da Virtuose e, cosa ancor più importante, non obbligatoriamente presenti all'interno di un'unica fonte di dati.

Per *profili* e tutte le altre parti del database non si sono trovati nuovi elementi e requisiti da introdurre.

A livello generale è stato deciso di rendere più comprensibile il ruolo svolto dalle varie strutture dati anche per persone non di lingua italiana; molte componenti dei dati, infatti, presentavano nella realizzazione proposta per la versione 2.0 nomi italiani e non tutti di facile comprensione "universale" e ciò , unito alla presenza di altri termini in inglese, portava anche ad avere poca uniformità: per questo è stato deciso di utilizzare la lingua inglese per ogni tag ed attributo della struttura XML dei dati.

### 2.2 Logica applicativa e relativa organizzazione

Anche per quanto riguarda la logica applicativa di Virtuose si è avuto un processo di espansione dei requisiti da soddisfare, arrivando a vere e proprie rivoluzioni rispetto alla versione precedente.

Come prima cosa si è voluta introdurre una vera possibilità di espandibilità e personalizzazione in modo da superare l'altissimo livello di staticità presente in Virtuose 1.0 andando al contempo ad aprire nuove possibilità di uso dell'applicazione. Per rispondere a questo requisito è stato deciso di suddividere la logia applicativa in unità funzionali distinte e in qualche modo indipendenti tra loro che avrebbero dovuto assolvere a compiti precisi.

Questo concetto di modularità era già presente nella prima release di Virtuose in cui si aveva una struttura a livelli, ognuno dedicato ad un ambito ben definito (database, creazione/gestione contenuti, interfaccia utente) e che prevedeva per ciascuno di essi la

presenza di blocchi funzionali specializzati ad assolvere determinati compiti; tutto questo però non era organizzato in modo da poter facilmente e trasparentemente inserire, sostituire ed eliminare un dato elemento funzionale portando perciò ad avere in pratica un sistema monolitico a basso indice di flessibilità e quindi difficilmente personalizzabile.

La fase di analisi di quali blocchi fondamentali avrebbero dovuto essere presenti per permettere di avere le stesse funzionalità base di Virtuose 1.0 ha portato alla definizione di moduli per: utenti, messaggi, profili, viste, conferenze e database.

Particolare attenzione è stata posta sul primo e l'ultimo di questi moduli, dato che oltre alla modularità ci si è resi conto di dover inserire nelle specifiche da seguire anche la possibilità di aver modo di sfruttare molteplici sistemi di gestione utenti in contemporanea (database SQL, database XML, LDAP, ... ), in modo da estendere il potenziale numero di fonti di dati per tali elementi fondamentali, e la capacità di supportare diverse tipi di database (SQL, XML, ...) senza che il resto dell'applicazione debba preoccuparsi di modificare la sintassi delle proprie richieste. Su quest'ultimo punto ci sono da fare alcune annotazioni:

- il fatto di poter utilizzare molteplici tecnologie per la base di dati (seppur avendone integrata ed attiva una sola alla volta) permette di introdurre un ulteriore elemento di flessibilità rispetto alla prima versione di Virtuose dato che l'applicazione era stata studiata e realizzata in modo da per poter sfruttare solo su un dbms relazionale basato su SQL, nel caso specifico PostgreSQL [PSQL].
- un solo tipo di sistema per la gestione dei dati può essere presente e utilizzabile in modo trasparente dal resto dei componenti, contrariamente a quanto fatto per gli utenti. Ciò comporterà la necessità in fase di installazione e configurazione di specificare il tipo di database (sql, xml, ...) e il relativo software di gestione utilizzato e quale modulo dovrà essere utilizzato dal sistema per interfacciarsi con esso.

La possibilità di sostituire un modulo qualsiasi con uno in grado di fornire gli stessi

servizi agli altri, oltre ovviamente alla possibilità di fornirne di ulteriori, è stata inserita anch'essa tra le nuove caratteristiche che dovrà possedere l'architettura di Virtuose andando così a porre un ulteriore elemento di flessibilità e modularità rispetto alla monoliticità dell'architettura di Virtuose 1.0.

Per rendere più agevole e meglio controllabile il flusso di informazioni in questa nuova struttura altamente modulare si è deciso di introdurre uno schema di comunicazione tra i moduli in grado di fornire uno standard abbastanza rigoroso da poter fornire precise linee guida e specifiche di implementazione e uso, lasciando al contempo ampi margini di flessibilità per poter supportare lo scambio di informazioni dai contenuti molteplici.

Tutto quanto evidenziato fino ad ora descrive lo strato più interno di Virtuose, in cui vengono eseguite le operazioni richieste dall'utente e quelle necessarie al controllo dell'intero sistema; oltre a ciò sono stati introdotti requisiti rivolti allo strato più esterno del sistema, quello relativo all'interfaccia utente e alla gestione grafica ad essa collegata.

Per quest'ultimo punto i requisiti non hanno subito notevoli modifiche rispetto a quelle già presenti nella versione precedente: è stato mantenuto il vincolo della separazione tra dati e presentazione imponendo che l'interfaccia utente venga gestita da un componente dedicato che dato un insieme di informazioni (XML) elaborate dall'applicazione e uno o più template grafici (XSLT), dopo opportune trasformazioni, porti alla creazione dell'interfaccia grafica necessaria per far interagire gli utenti con il sistema.

Per concludere l'elenco dei requisiti da soddisfare nella realizzazione della nuova versione di Virtuose, vi è da indicare che si è deciso di continuare ad usare tecnologie riconosciute come standard e solo software di tipo open source o free in senso generale. Anche questo punto permette di avere possibilità di personalizzazione più ampie in quanto consente eventuali modifiche più mirate e “profonde”.

# 3. Architettura di Virtuose 3.0

Dopo aver fornito nel capitolo precedente un'analisi dei requisiti e funzionalità da introdurre in questa nuova versione di Virtuose, mettendo anche in luce i principali punti di miglioramento rispetto a Virtuose 1.0 e al modello di database XML per l'eventuale versione 2.0, ora possiamo dare spazio alla presentazione di quello che è il risultato di tutta questa serie di ricerche, confronti e riflessioni.

Questo processo sarà diviso in due sezioni, la prima dedicata alla struttura base, formata dal nucleo applicativo, dalla struttura dei moduli, dal sistema di comunicazione tra questi ultimi e da come è stato deciso di gestire i contenuti , mentre la seconda vedrà una panoramica più dettagliata su come sono state organizzate le unità funzionali principali, in grado di gestire messaggi, conferenze, viste, profili, utenti e dati.

### *3.1. Struttura base*

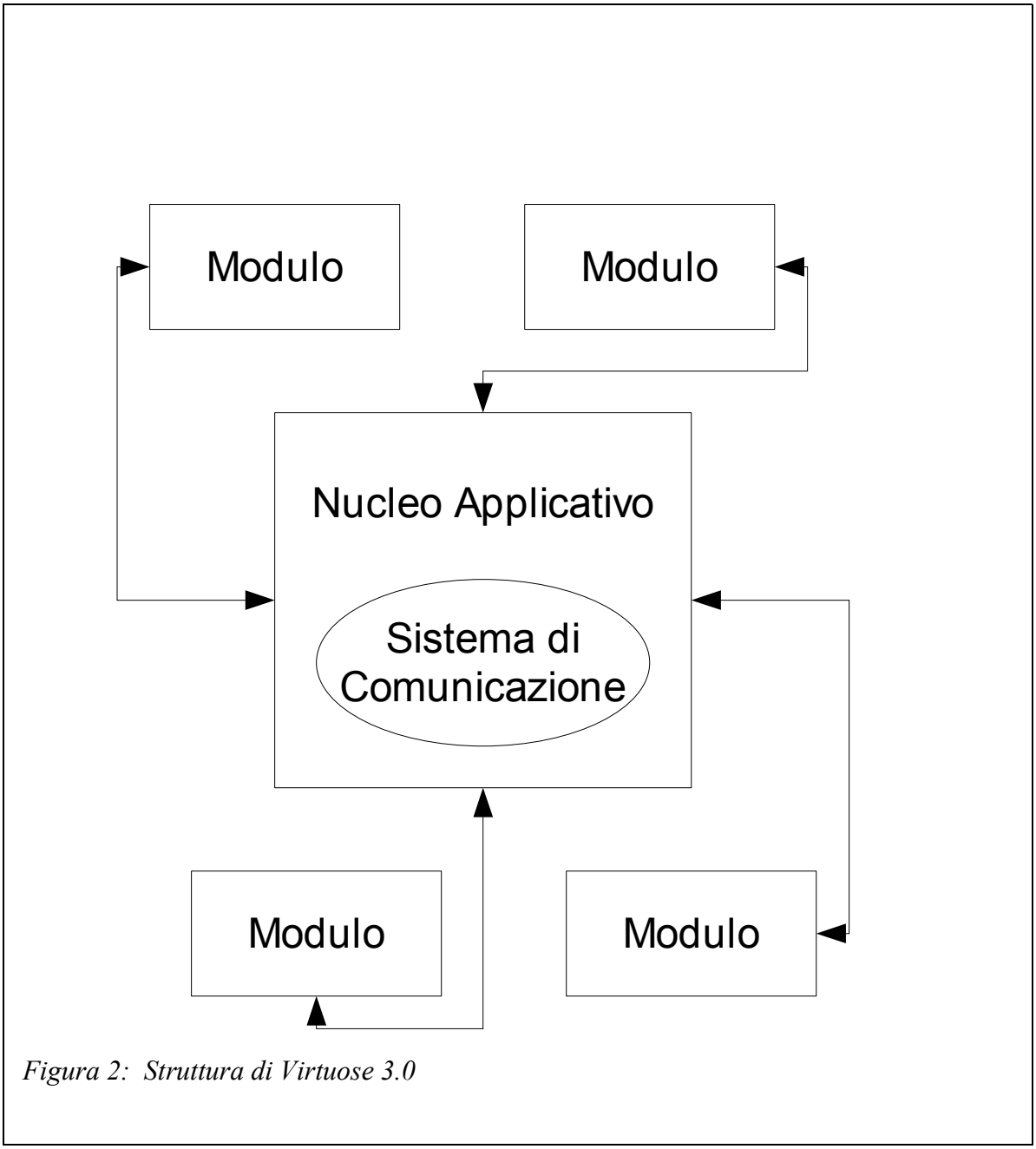


*Figura 2: Struttura di Virtuose 3.0*

L'organizzazione dei componenti nel nuovo Virtuose ha visto una modifica radicale nella sua conformazione ed infatti vi è stato il passaggio da una struttura a livelli sovrapposti ad una basata su un costrutto a stella (Figura 2). Con questa scelta si è voluto eliminare la stratificazione e la rigida disposizione dei moduli nel sistema, decentralizzandoli e rendendoli realmente indipendenti. Questo risultato si sarebbe potuto ottenere anche permettendo a tutti i componenti di scambiare direttamente tra loro, senza passare da un nodo centrale, le informazioni ed avendo in questo modo una struttura assimilabile ad un grafo totalmente connesso; con questa soluzione però si sarebbe avuta un'organizzazione più complessa e meno facilmente gestibile, oltre a porre l'obbligo per tutti i moduli di implementare al loro interno funzionalità di verifica, inizializzazione, uso e chiusura dei canali di comunicazione, compiti estranei a quelle che sono le loro funzionalità principali.

Per ottenere benefici ulteriori da questa riorganizzazione del codice è stato deciso di rendere il più leggero possibile, dal punto di vista della complessità e delle risorse richieste per il suo funzionamento, il componente centrale (nucleo applicativo, VK -Virtuose Kernel-) destinato a svolgere il compito di intermediario nelle comunicazioni tra moduli e nell'accesso di questi ultimi ai dati di configurazione globali del sistema; in questo modo il nucleo applicativo ha potuto essere progettato per i soli compiti di gestione della configurazione e per la ricezione di messaggi dalle varie unità di elaborazione, con conseguente instradamento degli stessi verso il destinatario corretto; in questo modo, in situazioni normali che non richiedono l'uso di mezzi di comunicazione diversi da quello di base, risulta pressoché ininfluente la richiesta di risorse per l'instradamento e recapito di richieste e risposte.

A diretto contatto con questo nucleo centrale altamente specializzato troviamo lo strumento utilizzato per lo scambio di informazioni tra i moduli, il sistema di comunicazione (VCS – Virtuose Communication System) che altro non è se non uno strumento dedicato per la gestione di semplice protocollo di comunicazione (VCP –

Virtuose Communication Protocol) basato sullo scambio di dati in formato XML, in cui è definita la modalità per invocare una qualsiasi delle funzionalità offerte da un modulo, studiato per offrire un insieme generico di funzionalità. Tali genericità e semplicità sono state usate come punto caratterizzante in quanto si era già deciso di porre le basi per poter inglobare il protocollo all'interno di un altro sistema di comunicazione, in modo da rendere più semplice le comunicazioni tra Virtuose nella sua forma basilare ed eventuali altri moduli o applicazioni non strettamente basati sulla sua architettura o direttamente accessibili dal nucleo applicativo.

I beneficiari diretti di questo metodo di comunicazione sono i moduli, le vere e proprie unità di elaborazione principali il cui compito è fornire funzionalità utilizzabili dagli altri componenti attivi nel sistema o accessibili da esso; grazie al fatto di avere una modalità standard attraverso la quale ricevere ed inviare messaggi, i moduli possono essere costruiti avendo come interfaccia verso l'esterno una piccola porzione di codice che andrà a determinare quale funzionalità è richiesta dal chiamante ed in base a ciò effettuerà le opportune elaborazioni che, una volta terminate, restituiranno i risultati sempre attraverso il canale lungo il quale è giunta la comunicazione. I moduli quindi possono essere visti come vere e proprie *black-box* il cui contenuto è totalmente nascosto mentre sono chiari i compiti svolti.

Questi sono in sintesi i componenti che sono alla base di Virtuose 3.0 e che formano l'architettura principale sulla quale si è poi costruito il prototipo vero e proprio del software per la gestione di contenuti e comunità virtuali. Come si può notare non è stato previsto a priori nessun livello specializzato in compiti particolari, ad esempio gestione di una base di dati e realizzazione dell'interfaccia utente, in quanto ogni modulo sarà realizzato in modo da essere specializzato in un dato insieme di funzionalità determinate dalle necessità individuate di volta in volta.

### 3.1.1. Nucleo applicativo

Come già emerso durante la stesura dei requisiti e nella presentazione globale dell'architettura di Virtuose il nucleo applicativo altro non è se non il nodo centrale per le comunicazione tra moduli e per l'accesso in sola lettura da parte di questi ultimi ai dati di configurazione del sistema; oltre a questi due compiti, orientati alla fornitura di servizi ad entità esterne a se stesso, in realtà ne è presente un terzo che va a porsi a monte di ogni altro: il processo di inizializzazione.

Attraverso questa funzionalità vengono predisposte tutte le strutture dati fondamentali per il corretto funzionamento del sistema; tale operazione iniziale è suddivisa in tre passi:

- lettura e memorizzazione del file di configurazione globale
- predisposizione del servizio per permettere l'accesso a tali dati da parte dei moduli o di altro codice applicativo
- predisposizione del protocollo di comunicazione di base.

Per quanto riguarda il primo punto sono fornite le funzioni base per cercare un particolare dato ed ottenerne il valore. Su come sono strutturati questi dati conviene dare qualche delucidazione.

Nel file di configurazione, anch'esso basato su XML, si possono riscontrare due sezioni, una dedicata ai moduli e alle informazioni a essi collegate e l'altra contenente le informazioni sulla modalità di installazione e configurazione (directory principale, numero di versione, directory contenente i template grafici, etc).

Entrando nello specifico sulle informazioni memorizzate su ciascun modulo abbiamo la presenza del nome, di un insieme di eventuali altri nomi simbolici (*alias*) per poter identificare lo stesso modulo attraverso molteplici denominazioni, di un indicatore il cui

scopo è indicare se il modulo è o meno residente nell'installazione locale di Virtuose, se può essere utilizzato su di esso un meccanismo di caching, eventuali parametri per l'inizializzazione e per finire un insieme di regole e informazioni per poter gestire e utilizzare su di esso protocolli di comunicazione diversi da quello di base fornito dal sistema.

Diamo ora qualche informazione aggiuntiva su come è strutturato il meccanismo per instradare le richieste giunte da un modulo e destinate ad un altro; anche in questo caso siamo di fronte ad una serie di passi, infatti abbiamo:

- ricezione della richiesta ed estrapolazione del modulo da contattare
- controllo dell'esistenza del modulo destinatario e scelta del protocollo di comunicazione da utilizzare
- uso del protocollo di comunicazione per l'invio del messaggio e per la ricezione della risposta ad esso correlata
- invio al chiamante della risposta ricevuta

Ora dovrebbe risultare più chiaro come tra i moduli non esista modo di dialogare direttamente senza passare dal nucleo applicativo e come quest'ultimo sia in effetti il vero punto chiave su cui si basa tutta la struttura di Virtuose per il proprio funzionamento; questa centralità, inoltre, pone l'obbligo di avere una implementazione il più possibile rapida, parca nelle risorse richieste e al tempo stesso solida ed in grado di sopportare carichi di lavoro gravosi in modo da non avere un effetto collo di bottiglia proprio in un punto così critico e vitale del sistema.

### 3.1.2. Sistema di comunicazione

Questo componente strutturale del sistema è quella che svolge, insieme al nucleo applicativo, i compiti fondamentali per permettere il corretto flusso di informazioni tra le varie parti del codice applicativo e per questo è, nella sua forma più semplice, inizializzato in fase di avvio, prima di avere reali necessità di comunicazione.

Data la sua natura di elemento fondamentale per il corretto funzionamento del sistema anche per esso è stata predisposta una struttura semplice, flessibile e per la gestione della quale non si necessita di grandi risorse, in modo da ridurre al minimo le possibilità di sovraccarico durante il suo utilizzo e al contempo massimizzare la sua facilità d'uso da parte dei moduli attraverso la mediazione del nucleo applicativo; vediamo di dare qualche ulteriore dettaglio.

Un modulo quando necessita di funzionalità "esterne" ad esso deve conoscere il nome, o un alias, con il quale il modulo richiesto è registrato nel sistema e deve inoltrare tale dato, unitamente al messaggio stesso da inviare, al nucleo applicativo il quale a questo punto delega il lavoro al sistema di comunicazione vero proprio. Giunti a questo punto si aprono due possibili strade, l'invio del messaggio lungo il canale standard o lungo uno dei canali alternativi sviluppati per incapsulare o estendere quello di base; la decisione è presa in funzione dei dati di configurazione specifici per il destinatario in questione.

Questa doppia opportunità si riflette nella struttura organizzativa del sistema di comunicazione, che è funzionalmente scisso in due parti di cui una integrata all'interno del nucleo applicativo (VK) mentre l'altra esterna ad esso.

La parte interna al VK è quella deputata a gestire le comunicazioni basate sul protocollo fondamentale pensato per Virtuose ed è stata integrata nella struttura in questo modo per fornire il massimo livello di prestazioni nelle azioni svolte in "locale" dato che per assolvere ai propri compiti necessita di dialogare con il nucleo applicativo per arrivare ad

avere caricamento ed esecuzione del modulo cui è indirizzato il messaggio.

Discorso diverso per il gestore della comunicazione non basato direttamente sul protocollo di comunicazione di base in quanto, in questo caso, non vi è nessuna struttura particolare già presente nel sistema ma tutto è delegato al componente integrato nel sistema, operazione a carico dell'amministratore, il cui riferimento è indicato nell'apposita sezione del file di configurazione generale. Per dare comunque delle basi e dei vincoli da rispettare è stata data una struttura di minima che questi elementi, il cui comportamento è quello di *plugin,* devono implementare. Tale organizzazione interna ricorda molto quella dei moduli, che sarà presentata più avanti, in quanto sarà totalmente incentrata intorno ad un unico punto chiave, il quale verrà utilizzato come tramite tra il nucleo applicativo cui è giunta la richiesta per l'apertura del canale di comunicazione e il modulo "remoto" da contattare e di cui recuperare e inoltrare all'indietro la risposta.

### 3.1.3. Moduli

I moduli sono organizzati in due sezioni distinte, una dedicata all'interazione con il nucleo applicativo per quanto riguarda le operazioni di gestione e l'altra orientata alla ricezione di messaggi attraverso il sistema di comunicazione di base di Virtuose; con tale suddivisione si è cercato di seguire anche in questo caso il concetto di separazione dei compiti e semplicità architetturale che è alla base dei requisiti principali da soddisfare individuati ad inizio lavoro e che ha avuto notevoli ripercussioni fino alla fine del processo di implementazione.

Vediamo ora di analizzare come queste due parti sono a loro volta internamente strutturate.

La sezione dedicata ai compiti di gestione prevede due funzionalità base messe a disposizione per l'interazione con il sistema, in particolar modo con il nucleo applicativo, di cui una specifica per l'inizializzazione del modulo e l'altra per la sua disattivazione. Attraverso la prima il sistema è in grado di avviare il modulo e di attivare le sue procedure di configurazione in modo da portarlo in uno stato di perfetta funzionalità, mentre utilizzando la seconda si attiva la procedura per la disattivazione.

La parte del modulo dedicata alla gestione delle comunicazioni attraverso l'uso del VCP è incentrata intorno ad un solo punto di accesso studiato per essere l'interfaccia attraverso la quale eseguire le operazioni di ricezione messaggi e invio dei risultati delle elaborazioni ad essi associate.

Ovviamente non è stato posto alcun vincolo sulla struttura che il modulo avrà al di fuori di queste sue parti appena presentate, né tanto meno ne sono stati imposti su come potrà essere organizzato il flusso di esecuzione correlato ad esse.

Inizialmente si era pensato di dare specifiche più stringenti sul come dovesse essere strutturato un modulo e in quest'ottica era stato portato avanti uno studio su quali

funzionalità comuni ogni modulo avesse dovuto avere. Durante questa primissima fase sono state individuate, sempre nell'ottica degli obiettivi e scopi specifici di un applicazione quale Virtuose, le funzionalità basilari di: avvio e configurazione, inserimento dati, lettura dati, modifica dati, cancellazione dati, ricerca sui dati, arresto.

Secondo l'ipotesi di lavoro iniziale queste capacità sarebbero dovute essere state accessibili come se fossero unità distinte, avendo per ciascuna di esse una struttura del messaggio perfettamente modellata sulle necessità del caso ma sempre e comunque rispettante le regole basilari emerse durante la progettazione del protocollo di comunicazione.

Oltre a queste funzioni comuni si era pensato di introdurre un'ulteriore possibilità per permettere di ricevere messaggi che andassero a richiedere operazioni non presenti tra quelle di base, ponendo così le basi per espandibilità e personalizzazione dei moduli.

Giunti ad avere questa prima bozza sull'organizzazione dei moduli si è visto come tale struttura fosse perfettamente adatta a gestire Virtuose nella sua forma di gestore di contenuti e di comunità virtuali ma come fosse al contempo troppo specializzata e portasse con se difficoltà nell'integrazione di componenti non strettamente rivolti a tali ambiti; uno qualsiasi di questi moduli in partenza estranei a Virtuose si sarebbe trovato a dover fingere di gestire comandi per lui privi di senso (inserimento dati,...) per poi gestire tutte le proprie funzionalità all'interno della struttura dedicata ai messaggi aggiuntivi o, cosa ancor peggiore, a dover usare l'infrastruttura pensata per i tipi di azione “naturali” per Virtuose per eseguire azioni fondamentalmente diverse da esse. Entrambe queste soluzioni avrebbero portato ad avere un uso non corretto o poco ottimizzato delle specifiche studiate per i moduli.

Dati questi problemi si è deciso di generalizzare e semplificare al massimo i vincoli e i principi cui i moduli devono far capo arrivando, come spiegato in precedenza, ad avere un solo punto di accesso al modulo per i messaggi e, di fatto, eliminando l'elenco delle funzionalità base.

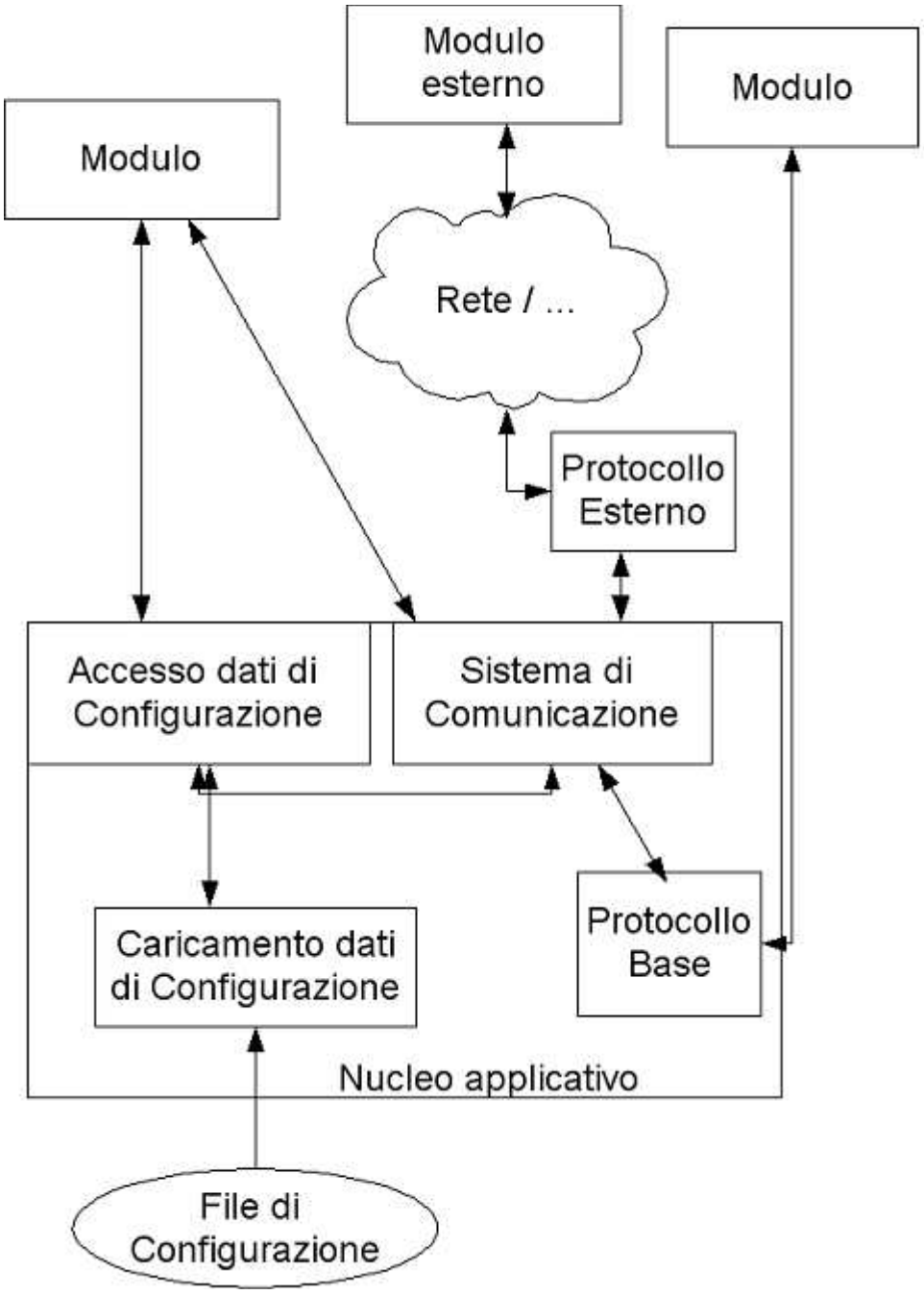


*Figura 3: Organizzazione moduli e comunicazione in Virtuose 3.0*

### 3.1.4. Gestione dei contenuti

Fino a questo punto è stata presentata l'architettura di Virtuose in quanto applicazione generica costituita da un insieme di moduli, dotati di struttura di base comune e che comunicano tramite messaggi inviati lungo un sistema di comunicazione, il tutto governato e gestito da un nucleo applicativo centrale.

Finora non si è specificato come sono organizzati i dati relativi ai contenuti gestibili dal sistema, vediamo quindi di far luce su questo aspetto tutt'altro che secondario.

Il processo di analisi per l'individuazione di come strutturare i contenuti e di come dar loro un'eventuale organizzazione è iniziato con il trovare i concetti fondamentali da utilizzare come fondamenta del sistema; avendo presenti i concetti di Conferenza e Messaggio così come sfruttati in Virtuose 1.0 è stato subito chiaro che affidare ad essi il compito di reggere la struttura contenutistica sarebbe stata una buona scelta. Ovviamente nella loro forma originale tali entità erano pensate per modellare un sistema basato su discussioni o aree tematiche e su messaggi a loro correlate, mentre nella nuova visione questo fatto deve essere superato arrivando ad avere un qualcosa in grado di gestire tali tipi di contenuti e al contempo qualsiasi altra tipologia di informazioni.

Con questo nuovo requisito Virtuose viene a trovarsi in una situazione tale da poter gestire tipologie di contenuti dei più disparati tipi, andando a superare il concetto di semplice applicazione per la gestione di comunità online basate sulla conversazione tramite messaggi testuali e arrivando a permettere l'uso di qualsiasi mezzo espressivo, integrando al contempo le possibilità di organizzazione delle informazioni tipiche dei software appartenenti alla categoria dei Content Management System (CMS).

Vediamo ora a quale forma si è giunti per la definizione di conferenze e messaggi come coppia di strumenti da usare in simbiosi.

***Conferenze***

Le conferenze giocano nel sistema il ruolo di aggregatori di messaggi, sia dal punto di vista puramente logico-strutturale sia da quello semantico; oltre a ciò vengono anche impiegate per fornire all'organizzazione dei contenuti una struttura gerarchica, non vincolata unicamente ad una "forma ad albero" ma con possibilità di estensione a "grafo". Questo è reso possibile dal fatto che ogni conferenza porterà con se informazioni circa quale è da considerare come proprio antecedente nella gerarchia e quale eventualmente è da intendersi come reale entità cui sta facendo riferimento, questo nel caso in cui sia un segnaposto (alias).

Attraverso i dati di parentela diretta e alias è possibile creare reti gerarchiche complesse, attraverso le quali creare raggruppamenti paralleli da dedicare, ad esempio, ad una organizzazione semantica; per avere un esempio visivo del come tutto ciò può essere messo in pratica vedere Figura 4.

Oltre ad avere nella propria struttura questo genere di informazioni, le conferenze possiedono anche riferimenti alle tipologie di contenuti (messaggi) che è possibile memorizzare al loro interno, questo per poter meglio controllarle e specializzarle in base alle esigenze da soddisfare; è da mettere in evidenza come quest'ultimo dato non sia ereditato dai discendenti della conferenza.

Per fornire poi un'altra possibilità di raccolta e organizzazione di questi contenitori di informazione è stato deciso di inserire la possibilità di creare dei gruppi trasversali facenti riferimento ad una data conferenza, ciò per permettere di avere strumenti di amministrazione in grado di agire automaticamente su molteplici di tali entità replicando le azioni eseguite su un solo rappresentante del gruppo.

Concludendo la panoramica sulla struttura che è stata decisa di fornire alle conferenze bisogna indicare che l'intero insieme di dati appartenenti a Virtuose e necessari al suo funzionamento è costruito sulla base di tale entità, i vari contenuti/informazioni sono memorizzati in messaggi a loro volta inseriti all'interno di conferenze dedicate a precisi

compiti; oltre a quelle standard ne esistono di fondamentali per il sistema, nelle quali sono memorizzati i dati sulle Viste, i Tipi di messaggio, i Profili, gli Utenti e altre due che sono quasi delle eccezioni alla regola, una "fantasma" che non contiene alcun messaggio e che serve a fornire il concetto di *conferenza nulla* e un'altra che costituisce la radice principale della struttura a cui sono legate tutte le altre.

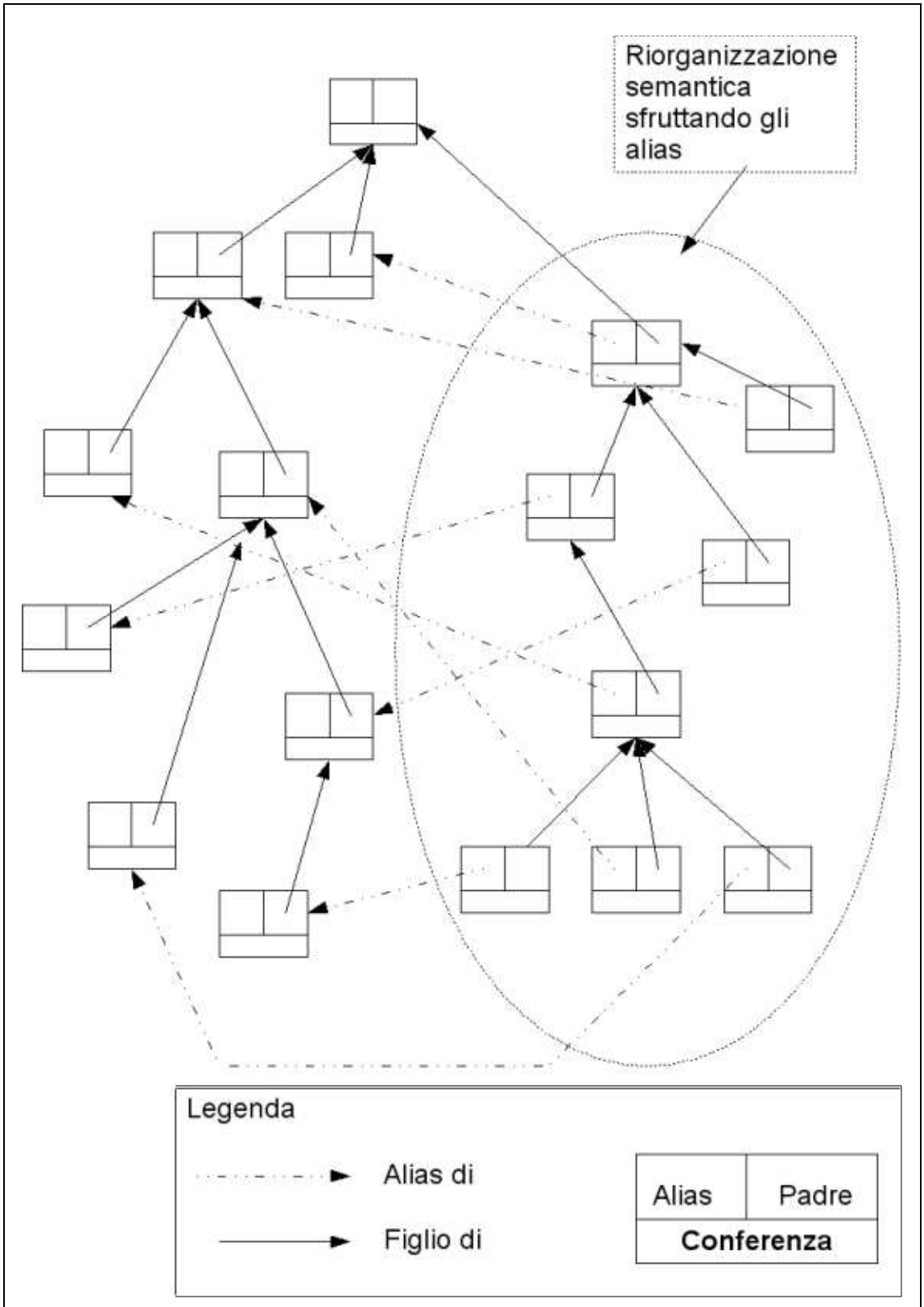


*Figura 4: Esempio di organizzazione  delle Conferenze*

***Messaggi***

Nell'architettura di Virtuose ogni elemento destinato a rappresentare un contenuto è memorizzato sotto forma di messaggio la cui struttura è in parte predefinita e costante e in parte dipendente dal tipo del contenuto stesso. Questa organizzazione è rimasta pressoché invariata se confrontata con quella della prima versione dell'applicazione in quanto già allora era emersa la volontà di far convergere l'idea di "dato" verso quella di messaggio; la caratteristica interessante di questo *fornitore di contenuti* è la modalità con la quale si crea o modifica l'insieme di specifiche volte a descrivere la struttura dei contenuti stessi.

Questo processo è suddiviso in due fasi, quella di costruzione vera e propria delle regole per la definizione del tipo e quella di assegnamento dello stesso ad uno o più messaggi.

Partiamo da quest'ultima operazione dato che è la più semplice e meno innovativa in quanto si tratta solo di inserire l'identificativo corretto nella sezione ad hoc della struttura del messaggio.

Per la creazione e definizione del tipo di messaggio, invece, si sfruttano le potenzialità messe a disposizione dall' XML-Schema [VDV02] integrandole con i concetti e le idee su cui poggia la struttura dei dati.

Innanzitutto va detto che il sistema prevede dei tipi di messaggio predefiniti tra i quali troviamo quelli relativi alle viste, ai profili, agli utenti e quello generico che, non ponendo vincoli su struttura e valori dei dati di alcun tipo e lasciando quindi massima libertà, permette di avere una base sulla quale poggiare tutti gli altri; questa necessità viene dal fatto che anche la definizione di tipo è memorizzata come messaggio e quindi deve avere un'indicazione su come deve essere il suo contenuto per essere valido e dunque usandone una che implica sempre e comunque tale valutazione di adesione alla specifica come "corretta" si aggira il problema senza infrangere alcuna regola.

Per quanto riguarda la creazione di un nuovo tipo di messaggio, o la modifica di uno preesistente, si usa il linguaggio XML-Schema, come accennato in precedenza, attraverso il quale è possibile definire delle regole che permettano di decidere se un dato frammento di codice XML segue o meno le specifiche da noi volute (tale processo viene chiamato *validazione*) e che verranno archiviate come normalissimo contenuto di un messaggio.

Va precisato che le regole di validazione memorizzate per il tipo associato ad un dato messaggio sono valide e utilizzabili solamente sulla parte della struttura del messaggio stesso non prefissata a priori e comune a tutti i messaggi, in pratica sono applicabili al solo contenuto e non alle informazioni di posizionamento nella gerarchia generale e a quelle utili al sistema.

Questo tipo di organizzazione per la gestione delle regole di validazione dei contenuti può essere visto come una struttura ricorsiva in cui l'elemento più interno è l'origine di tutti gli altri,oltre a presentare la caratteristica di essere autoreferenziale

Nella prossima figura (Figura 5) vediamo rappresentato graficamente questo aspetto del sistema.

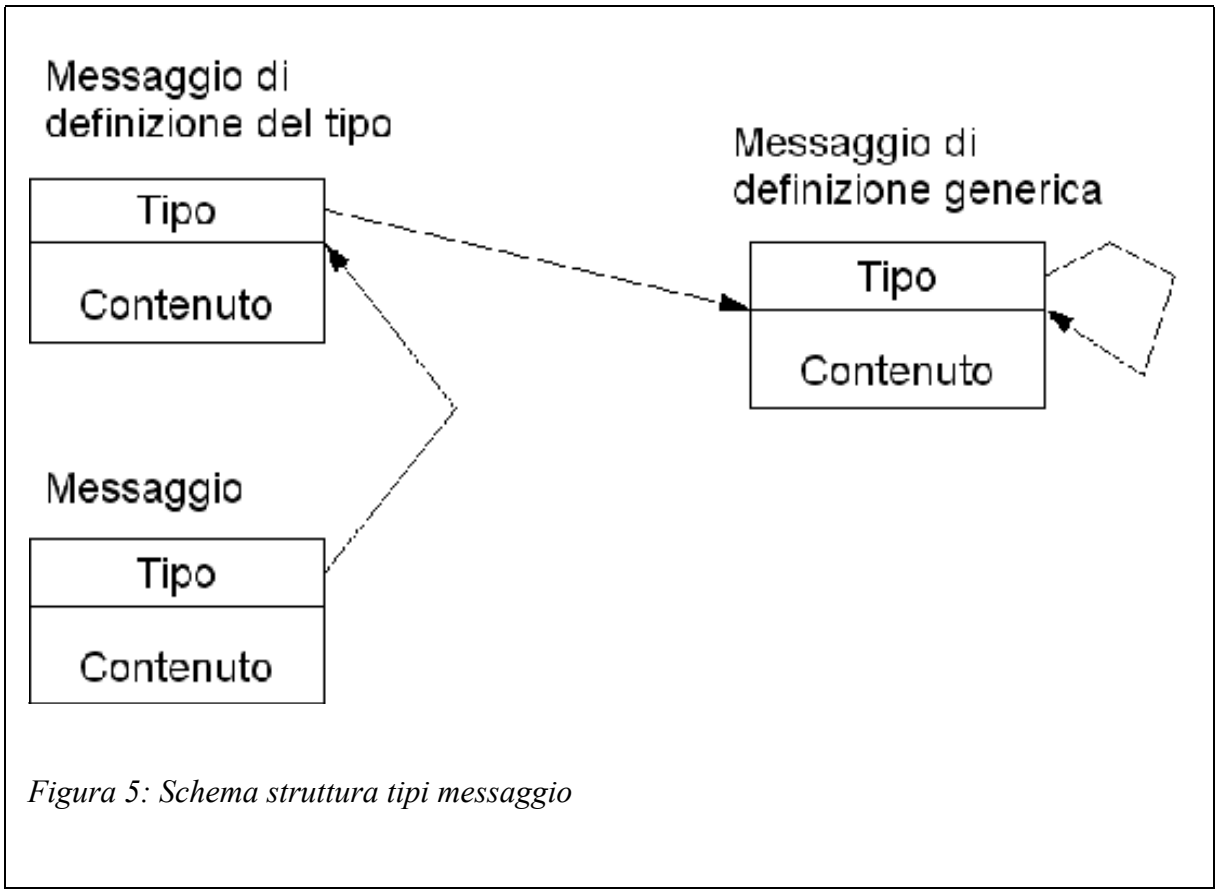


*Figura 5: Schema struttura tipi messaggio*

### *3.2. Principali Moduli e relative funzionalità*

Fino a questo punto è stata trattata l'organizzazione della nuova versione di Virtuose solo da un punto di vista piuttosto generico, sebbene siano state fornite informazioni in alcuni casi su precise componenti non si è mai entrati nel dettaglio di alcuno dei moduli fondamentali per il sistema.

Per la gestione dei contenuti e delle attività principali di una comunità virtuale sono stati identificate come fondamentali le funzionalità messe a disposizione per la gestione dei messaggi, delle conferenze, degli utenti, dei profili, delle viste e dell'immagazzinamento dei dati contenute nei rispettivi moduli: Messages, Conferences, Users, Profiles, Views e Storage.

Vediamo ora quale funzionalità sono state integrate nella struttura di ciascuno di essi e quali eventuali peculiarità sono presenti.

### 3.2.1. Messages

Questo modulo è quello deputato a fornire le funzionalità necessarie a manipolare i messaggi presenti nel sistema. La sua organizzazione, come quella di tutti i moduli, è incentrata intorno ad un punto di ingresso per le richieste provenienti dal sistema attraverso il protocollo di comunicazione fondamentale, dalla quale vengono richiamate le diverse sezioni per l'esecuzione dei compiti relativi all'interrogazione pervenuta.

Vediamo ora quali sono le funzionalità offerte dal modulo:

- lettura: dato l'identificativo univoco permette di estrarre dal database l'intera struttura, dati di sistema e contenuto, di un messaggio
- inserimento: permette di inserire un nuovo messaggio avente i dati parametri di sistema e contenuto. Da notare che non è presente direttamente a questo livello la validazione del contenuto in funzione del tipo fornito tra i parametri, è stata fatta questa scelta per non appesantire questa funzionalità e pertanto ciò sarà a carico del chiamante.
- aggiornamento: permette di modificare le informazioni e il contenuto associati ad un dato messaggio, anche in questo caso come nel precedente non ha luogo alcun meccanismo di validazione
- eliminazione: permette di eliminare un messaggio dal database
- ricerca: permette di trovare messaggi corrispondenti ai criteri ricevuti come parametri
- estrazione dei messaggi di un dato utente: funzionalità che permette di tutti i messaggi presenti nel sistema che sono stati indicati come riferiti all'utente il cui identificativo è stato indicato tra i dati della richiesta
- estrazione dei messaggi di un dato tipo: partendo dall'identificativo di un tipo di

messaggio permette di ottenere tutti i messaggi memorizzati nel database che sono indicati come facenti riferimento ad esso

- estrazione dei messaggi di un dato tipo e appartenenti ad uno specificato utente: con questa funzionalità vengono fuse le capacità delle due precedenti, si ha quindi una ricerca ed estrazione dei messaggi di un dato tipo e appartenenti ad un utente specifico
- estrazione del contenuto: dato l'identificativo del messaggio si ottiene il suo contenuto in forma grezza, esattamente come è stato inserito nel database e senza nessuna manipolazione su di esso
- estrazione dei messaggi direttamente correlati con un dato messaggio: permette di ottenere i dati completi (struttura base e contenuto) dei messaggi per i quali è stato indicato in fase di creazione/modifica come riferimento di antenato diretto nella scala gerarchica quello indicato
- estrazione dei messaggi secondo una schema: con questa funzionalità si offre la possibilità di estrarre un messaggio e tutti quelli facenti riferimento, diretto o meno, ad esso seguendo una modalità ben precisa che definisce le regole per effettuare la scelta. Queste possibili tipologie di estrazione sono:
  - *thread* per avere il messaggio indicato e tutti quelli riferiti direttamente ad esso (utile per visualizzazione sotto forma di lista)
  - *nested* per avere il messaggio principale e l'insieme di quelli ad esso collegati direttamente e non (da usare principalmente per una anteprima con informazioni gerarchiche)
  - *extended* per avere gli stessi risultati della modalità *nested* ma orientati ad una visualizzazione estesa
  - *plain_list* utilizzabile per avere l'elenco di tutti i messaggi collegati ad uno dato

(sfruttabile per una visualizzazione tramite anteprima degli stessi e completa per quello principale).

- estrazione identificativo conferenza di appartenenza: quest'ultima funzionalità permette di ottenere il riferimento alla conferenza nella quale è contenuto il messaggio su cui si sta operando.

### 3.2.2. Conferences

Questo modulo, strutturato secondo le linee guida generali già indicate più volte, è quello incaricato di fornire al sistema funzionalità varie circa la gestione e manipolazione delle conferenze, ecco quali:

- lettura: estrae dal database l'intera struttura dati della conferenza corrispondente all'identificativo fornito, compresi tutti i messaggi in essa contenuti
- inserimento: inserisce una nuova conferenza avente le caratteristiche fornite attraverso i parametri della richiesta
- aggiornamento: permette la modifica dei dati associati alla conferenza indicata tramite il suo identificativo
- eliminazione: esegue la rimozione dal database della conferenza avente l'identificativo indicato, nel processo devono essere eliminati anche i messaggi in essa contenuti
- ricerca: estrae le conferenze che sono conformi ai parametri ricevuti
- estrazione degli identificativi messaggi: data una conferenza, sotto forma di suo identificativo, permette di ottenere tutti gli identificativi dei messaggi in essa contenuti, senza applicare ad essi alcun criterio di selezione aggiuntivo
- risoluzione degli alias: dal momento che è stato aggiunto ai dati delle conferenze anche uno destinato ad essere usato per la creazione di alias è stata introdotta questa funzionalità che permette di risalire dalla conferenza "segnaposto" a quella reale, attraversando tutti gli eventuali altri alias intermedi
- estrazione degli identificativi dei messaggi di un tipo dato: permette di ottenere gli identificativi dei messaggi definiti come corrispondenti al tipo fornito come parametro presenti all'interno della conferenza indicata

- estrazione degli identificativi delle conferenze discendenti da quella data: dal momento che nelle conferenze è possibile introdurre il riferimento ad un'altra di esse considerata come ascendente diretto nella gerarchia è stato necessario introdurre questa funzionalità il cui scopo è quello di estrarre tutti gli identificativi dei discendenti diretti di quella indicata, in pratica si esegue il procedimento inverso di quello inglobato nella struttura dati
- estrazione delle informazioni: con questa funzionalità viene fornita la possibilità di estrarre dalla conferenza, indicata attraverso il suo identificativo, l'insieme dei suoi dati informativi senza andare a ricavare anche quelli circa i messaggi in essa contenuti. In questo caso abbiamo un comportamento non del tutto "diretto", prima di inviare al richiedente le informazioni richieste è stato deciso di inserire un'ulteriore passo dipendente dalla funzionalità descritta precedentemente a questa ed infatti se la conferenza richiesta è in realtà un alias per un altra si effettua una riscrittura dei dati estratti in prima battuta usando quelli della conferenza "reale" al posto di quelli della segnaposto, lasciando inalterati solo quelli relativi al nome e al "padre" gerarchico.
- estrazione identificativi dei messaggi di vari tipi: questa funzionalità è analoga ad una di quelle mostrate in precedenza ma ad essa aggiunge la possibilità di indicare più tipi di messaggi sui quali basare i criteri di estrazione dalla conferenza in questione
- estrazione contenuto filtrato: questa funzionalità permette di estrarre una conferenza e i messaggi in essa memorizzati secondo uno schema preciso, un po' come accadeva con l'analoga funzionalità del modulo Messages. Le possibili metodologie di estrazione sono:
  - *thread* per avere solo un'anteprima dei messaggi che non fanno riferimento diretto ad altri (un esempio di messaggio di questo tipo è quello di apertura di una discussione)

- *nested* per avere un'anteprima di tutti i messaggi da usare per una visualizzazione in grado di dare informazioni circa la gerarchia esistente tra gli stessi
- *extended* per ottenere tutti i messaggi appartenenti alla conferenza e visualizzarli interamente dando le eventuali informazioni sulla gerarchia esistente
- *plain_list* per ottenere un'anteprima di tutti i messaggi senza informazioni gerarchiche.

- estrazione degli identificativi dei tipi di messaggio validi: permette di estrarre da una data conferenza gli identificativi dei tipi di messaggio che possono essere inseriti al suo interno
- estrazione degli identificati conferenze appartenenti ad un gruppo: data la presenza di possibili aggregati di conferenze basati su criteri non di tipo gerarchico è stata inserita la possibilità di ottenere un elenco dei membri di tali insiemi sotto forma di elenco di identificativi.

Come nota generale sulle funzionalità previste dal modulo Conferences bisogna indicare come si abbia sempre il processo di risoluzione degli alias e relative azioni collegate ovunque si abbia la necessità di ottenere dati appartenenti alla sezione "informativa" della struttura dati della conferenza memorizzata nel database.

### 3.2.3. Users

Questo modulo fornisce tutte le funzionalità strettamente legate alla gestione utenti, siano essi memorizzati nel database interno dell'applicazione o in fonti esterne. Vediamo innanzitutto quali sono le azioni permesse:

- login: permette di autenticare l'utente in modo da farlo riconoscere dal sistema differenziandolo da quelli anonimi, in alcuni casi questa azione richiederà l'esecuzione di altre funzionalità
- logout: permette di far tornare anonimo un utente rimuovendo dai dati della sessione d'utilizzo in corso qualsiasi riferimento alle sue credenziali, in certi casi potrebbe prevedere l'uso di funzionalità accessorie
- inserimento: permette di inserire un nuovo utente avente le credenziali specificate, di norma questa funzionalità è disponibile solo per quanto riguarda azioni destinate ad intervenire sul database generale di Virtuose
- eliminazione: permette i eliminare i dati relativi ad un utente, si applicano le stesse limitazioni della funzionalità precedente
- aggiornamento: da modo di modificare i dati associati all'utente indicato, anche in questo caso sono presenti limitazioni come per la funzionalità di inserimento
- lettura: permette, in base all'identificativo dell'utente, di estrarre i dati ad esso associati
- ricerca: questa funzionalità permette di estrarre tutti gli utenti che corrispondono ai criteri di selezione indicati

Questo insieme di operazioni è quello che può completamente essere sfruttato sui dati presenti nel database gestito dall'applicazione mentre per quelli al di fuori di esso non sarà, in genere, totalmente utilizzabile in quanto non è detto si abbiano permessi sufficienti per eseguire modifiche sulle informazioni gestite da altre applicazioni.

Tale differenza di comportamento è presente anche nella struttura stessa del modulo in quanto al suo interno trovano posto sia l'integrazione di tutte le funzionalità volte a permettere la gestione dell'utenza completamente gestita "in locale" sia un meccanismo per poterle eseguire su piattaforme esterne. Per permettere quest'ultima possibilità è stato creato un sistema basato su plugin, ognuno dei quali con al proprio interno la capacità di effettuare un sottoinsieme, o tutte, le funzionalità viste in precedenza e disponibili per la gestione utenti; ovviamente per una gestione ottimale è stato previsto un meccanismo di sincronizzazione minimale tra il database proprio di Virtuose e l'applicazione esterna in modo da avere internamente disponibili per ogni utente un certo numero di informazioni di base (email,identificativo,plugin da utilizzare,...) necessarie al corretto funzionamento del sistema.

Tutta questa organizzazione basata su componenti da utilizzare secondo necessità è completamente nascosta dal modulo Users al resto del sistema, ogni richiesta inviata al modulo è completamente indipendente da dove verrà estratto fisicamente il dato su cui lavorare e dai risultati dell'elaborazione; le uniche informazione riguardanti l'esistenza di componenti aggiuntivi per la gestione utenti sono ritrovabili nel database, dovo sono associate all'utente, e nel file di configurazione globale, dove sono utilizzate per indicare quali plugin utilizzare.

In Figura 6 ecco come appare questa organizzazione per la gestione utenti.

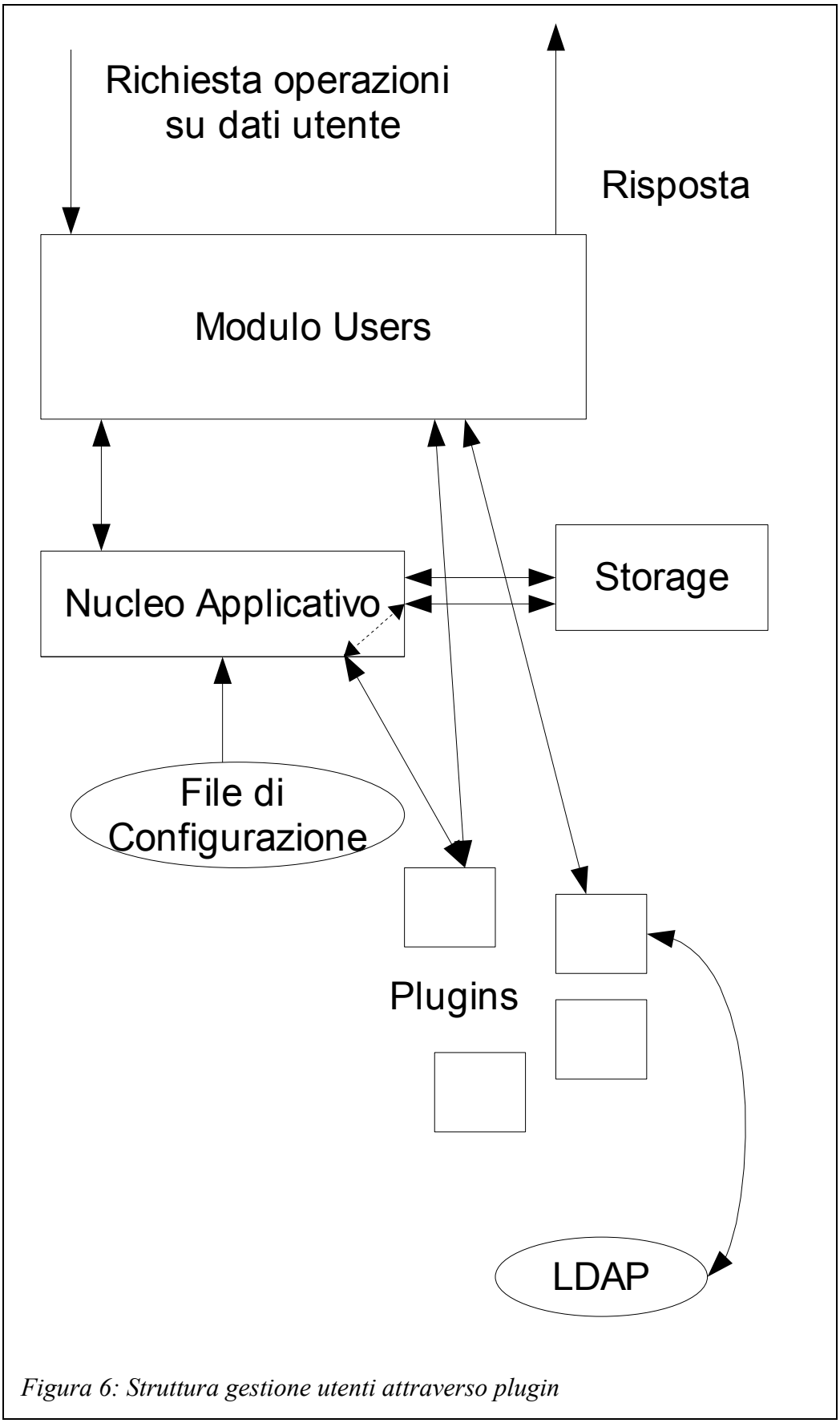


*Figura 6: Struttura gestione utenti attraverso plugin*

### 2.2.4 Profiles

Sfruttando questo componente è possibile agire sui profili, strutture dati utilizzate per assegnare ruoli agli utenti e molto legate anche alle viste. Le possibilità d'azione sono definite dalle seguenti funzionalità offerte:

- lettura: permette di ottenere i dati relativi al profilo indicato nella richiesta attraverso il suo identificativo
- inserimento: consente di inserire un nuovo profilo avete per informazioni associate quelle indicate
- aggiornamento: offre la possibilità di modificare i dati di un profilo
- eliminazione: permette di eliminare un profilo, identificato tramite il suo identificativo, dal database
- ricerca: dati i parametri di ricerca inviati nella richiesta offre la possibilità di estrarre i profili che soddisfano tali criteri di scelta
- estrazione delle viste associate: permette di ottenere gli identificativi di tutte le viste associate al profilo
- estrazione dei profili associati ad un utente: fornisce accesso all'elenco dei profili associati all'utente corrispondente all'identificativo fornito come parametro della comunicazione, i dati estratti comprendono ogni informazione associata ai profili
- estrazione dell'identificativo del profilo base: fornisce il riferimento al profilo base generalmente associato ad utenti del sistema senza particolari compiti o ruoli
- estrazione dell'identificativo del profilo anonimo: estrae l'identificativo del profilo associato agli utenti che non hanno ancora effettuato l'autenticazione

### 3.2.5 Views

Questo modulo si occupa di fornire tutta la serie di possibili azioni effettuabili sulle viste e per questo all'interno della struttura del nostro tipo di sistema per la gestione di contenuti e comunità virtuali risulta essere un componente dei più delicati in quanto il grosso dei meccanismi di sicurezza è incentrato sulle sue capacità. Vediamo ora quali sono le azioni messe a disposizione da questo modulo:

- lettura: offre la possibilità di ottenere l'intera struttura dati di una vista
- inserimento: permette di inserire una nuova vista con le caratteristiche inviate come parametri della richiesta
- aggiornamento: consente di modificare i dati preesistenti della vista corrispondente all'identificatore fornito sostituendoli con quelli inviati insieme alla richiesta
- cancellazione: permette di eliminare una vista
- ricerca: permette di estrarre dal database tutte le viste corrispondenti ai criteri specificati
- estrazione dei permessi su una conferenza: da modo di estrarre tutti i permessi e le informazioni di carattere grafico-stilistico relative ad una data conferenza in base alla vista specificata
- estrazione dei permessi su un tipo di messaggio in una conferenza: permette di ottenere quali permessi sono associati ad un dato tipo di messaggio nella conferenza indicata per la vista in analisi
- estrazione degli stili grafici per un tipo di messaggio in una conferenza: da modo di estrarre quali informazioni stilistico-grafiche sono associate al tipo di messaggio indicato quando interno alla conferenza indicata, il tutto sempre in funzione della vista sulla quale si sta effettuando l'interrogazione
- estrazione dei tipi di messaggio per una conferenza: da modo di ottenere tutte le

informazioni, sia permessi che grafica, dei tipi di messaggio sui quali la vista indicata offre informazioni relativamente alla conferenza specificata nella richiesta

- estrazione della lista delle sotto-conferenze visibili: estrae i dati relativi all'aspetto grafico e all'identificativo delle conferenze che sono state indicate come discendenti da visualizzare di una data conferenza nella vista in analisi
- estrazione della lista dei tipi di messaggio visibili in una conferenza: esegue l'estrazione degli identificativi dei tipi di messaggio inseribili in una data conferenza che secondo la vista specificata sono visibili (permesso di visione impostato a "true").

### 3.2.6 Storage

Fin qui abbiamo visto moduli con al loro interno funzionalità ben precise e indipendenti dalla configurazione del sistema, ciò non vale per quello dedicato alle interazioni con il database in quanto non sono previste a priori o direttamente inserite nel modulo stesso le capacità di interazione con i dati.

Come se ciò non bastasse ad identificare come "anomalo" il componente dedicato a fare da interfaccia tra il sistema e la fonte dei dati si ha anche che, in effetti, un modulo "Storage" vero e proprio non esiste in quanto tale identificativo sarà solo uno degli alias usati dalla versione specializzata e realizzata ad hoc per utilizzare il tipo di database scelto: per ogni database utilizzabile esisterà un modulo specifico. L'uso di questa struttura e organizzazione particolari è nato dal bisogno di fornire il massimo livello di flessibilità per quanto riguarda la scelta della tecnologia adibita a gestire la base di dati e perciò si è eliminato tutto ciò che potesse porre ostacoli o vincoli ulteriori.

Nonostante queste premesse per la gestione del database non vengono violati prerequisiti o specifiche fondamentali per la struttura dei moduli e restano validi i punti di accesso e le modalità di gestione delle comunicazioni attraverso il protocollo basilare visti in precedenza; vediamo ora come funziona questa parte del sistema.

Quando una parte del sistema tenta di aprire un canale di dialogo con il modulo Storage il nucleo applicativo riconosce il destinatario reale della comunicazione, attraverso i dati ottenuti dal file di configurazione ed in particolare attraverso la sezione *alias* presente nei dati relativi al modulo che andrà ad agire come tramite con il database, e operando di conseguenza invia la richiesta ricevuta ad esso che provvederà ad eseguire le operazioni del caso.

In questo modo abbiamo visto come, nonostante la mancanza di un modulo generico vero e proprio per fornire accesso ai dati, il sistema sia in grado di funzionare in modo

corretto nascondendo completamente questa particolare situazione ai vari componenti interessati ai servizi del database.

Ma come è possibile eseguire le azioni sui dati se il modulo non ha al suo interno nessuna funzionalità specifica oltre a quella per permettere di ricevere richieste ed inviare le risposte?

La risposta a questa domanda è semplice: attraverso l'uso di componenti esterni (plugin) che vanno ad integrare il modulo stesso, vediamo come.

Quando il modulo che agisce da "Storage" riceve una richiesta determina, in base all'azione da eseguire, su quale entità deve operare e quindi quale dei plugin presenti caricare; dopo aver fatto questa operazione sfrutta le funzionalità offerte da quest'ultimo e ricava come andare ad interagire nello specifico con il database.

Ovviamente per poter dare una certa stabilità all'architettura sono stati posti vincoli sul come devono essere organizzati questi componenti esterni ed usati dal modulo del database, ma anche in questo caso si è scelta la massima semplicità ed infatti essi offrono una sola unità applicativa la quale si occupa di analizzare i dati forniti dal modulo Storage e, in base ad essi, di restituire allo stesso lo stretto necessario per poter soddisfare la richiesta ricevuta.

Uno schema di come si presenta l'organizzazione della parte del sistema deputata a fare da interfaccia con il database è visibile in Figura 7.

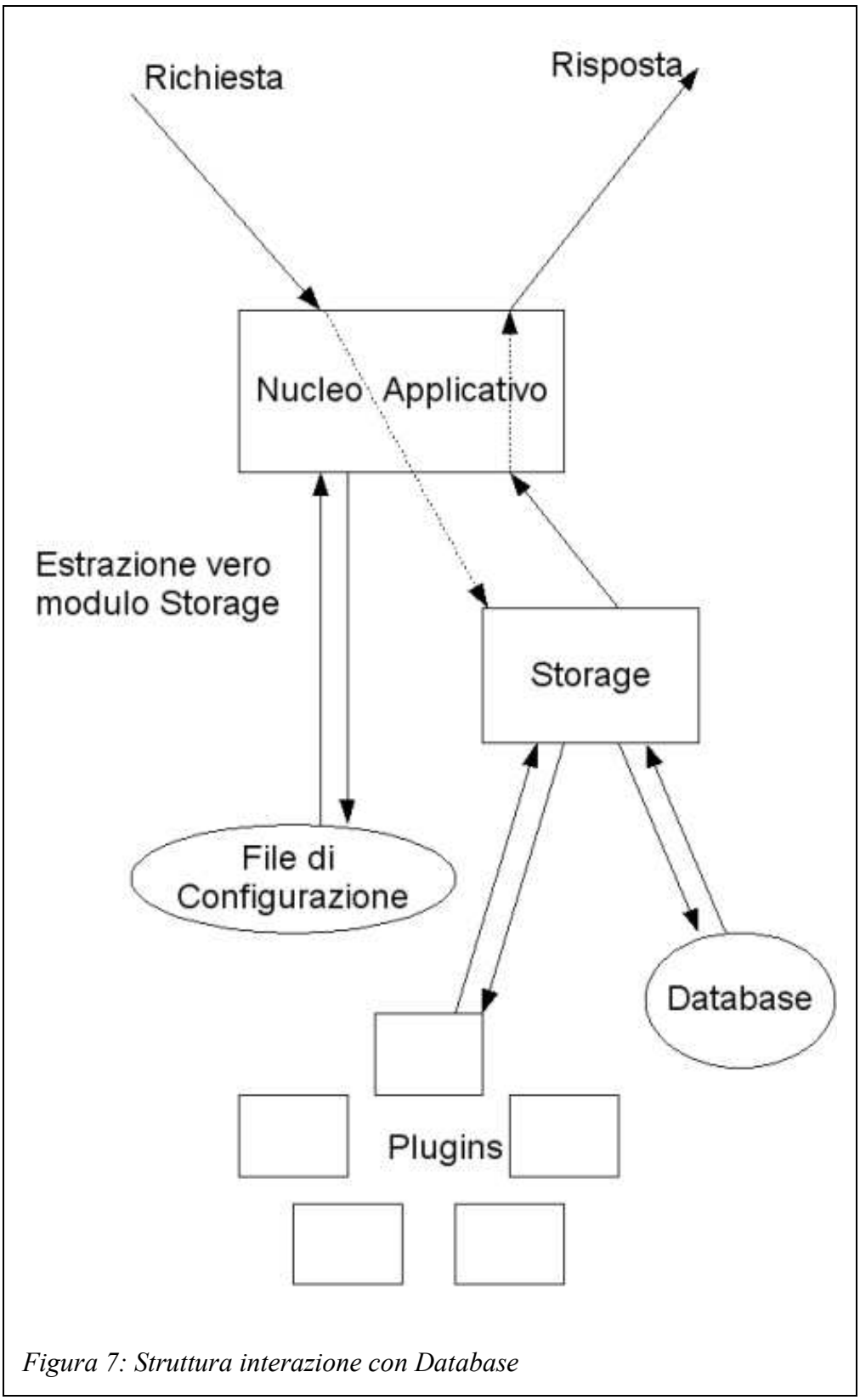


*Figura 7: Struttura interazione con Database*

### *3.3 Visualizzazione Conferenza*

Diamo ora un rapido sguardo al processo che porta ad avere una rappresentazione visiva di una conferenza, di cui è stato dato l'identificativo, pronta per essere fornita all'utente secondo la vista specificata:

- inizializzazione del gestore del layout, usato per caricare i documenti XSLT per le trasformazioni e per applicarli ai dati xml, appropriato in base indicazioni ricevute
- lettura dei parametri della richiesta (identificativo della vista e della conferenza)
- invio messaggio al modulo Conferences per ottenere il contenuto della conferenza (sottoconferenze e messaggi) in base all'identificativo di quest'ultima e della vista da utilizzare. La vista è necessaria per avere informazioni sui tipi di messaggio da considerare, sulla modalità di estrazione e sull'elenco delle sottoconferenze
- ricezione xml con la rappresentazione grezza dei dati
- invio messaggio al modulo Conferences per ottenere un elenco degli identificativi dei tipi di messaggio validi per la conferenza da visualizzare
- ricezione xml con l'elenco dei tipi di messaggio
- per ogni identificativo di tipo di messaggio ottenuto dalla chiamata precedente:
  - invio messaggio al modulo Views per ottenere, in base alla vista corrente, quale documento XSLT utilizzare per la formattazione dei messaggi di tale tipologia
  - caricamento e personalizzazione del documento XSLT e memorizzazione di tale insieme di dati
- inizializzazione del documento xml destinato a contenere il risultato del processo di creazione dell'interfaccia utente
- per ogni messaggio presente nell'xml contenente i dati grezzi della conferenza:
  - esecuzione della trasformazione XSLT sui dati del messaggio in base al foglio di

stile caricato in precedenza per il tipo del messaggio in esame

- memorizzazione del risultato nel documento xml contenente il risultato finale delle trasformazioni

- per ogni sottoconferenza presente nell'xml al cui interno sono presenti i dati non formattati della conferenza:
  - caricamento del documento XSLT indicato per la sottoconferenza in questione
  - esecuzione della trasformazione XSLT sui dati della sottoconferenza in esame in base al foglio di stile caricato al passo precedente
  - memorizzazione del risultato nel documento xml contenente il risultato finale delle trasformazioni
- inserimento dei dati generali della conferenza da visualizzare nell'xml utilizzato per tener traccia dei risultati dei processi di trasformazione su messaggi e sottoconferenze
- caricamento documento XSLT per la creazione degli ultimi dettagli stilistici globali da applicare a quanto creato fino ad ora
- esecuzione trasformazione finale
- invio all'utente del risultato

### *3.4 Visualizzazione Messaggio*

Vediamo adesso quale è l'insieme di azioni che dato un identificativo di un messaggio ed una vista porta alla creazione di quanto verrà presentato all'utente:

- inizializzazione del gestore del layout, usato per caricare i documenti XSLT per le trasformazioni e per applicarli ai dati xml, appropriato in base indicazioni ricevute
- lettura dei parametri della richiesta (identificativo della vista e del messaggio)
- invio richiesta al modulo Messages per ottenere l'identificativo della conferenza in cui è inserito il messaggio in esame
- ricezione xml con il dato richiesto
- invio richiesta al modulo Messages per ottenere i dati memorizzati nel messaggio da visualizzare
- ricezione xml con le informazioni richieste
- invio messaggio al modulo Views per ottenere l'informazione circa quale documento XSLT è da utilizzare per il messaggio in esame
- ricezione xml contenente i dati relativi al file da usare
- invio richiesta al modulo Messages per ottenere la lista di informazioni necessarie alla visualizzazione del messaggio, tra queste ci saranno anche tutti i dati degli eventuali altri messaggi ad esso correlati e che dovranno essere parte della presentazione
- ricezione xml con rappresentazione grezza dei dati
- per ogni messaggio estratto:
  - richiesta al modulo Views per ottenere i dati del documento XSLT da utilizzare
  - ricezione dati xml contenenti l'informazione
  - caricamento del documento XSLT ricavato

- esecuzione della trasformazione XSLT sui dati del messaggio in esame in base al foglio di stile caricato al passo precedente
- memorizzazione del risultato nel documento xml contenente il risultato finale delle trasformazioni

- invio all'utente del risultato complessivo delle trasformazioni

# 4. Implementazione

Fino ad ora si è discusso di quali limiti e problemi sono stati individuati nella prima versione di Virtuose e nell'evoluzione del suo database e di come si sia cercato di superarli, giungendo infine a mostrare come tutto ciò abbia influito sull'organizzazione della struttura della terza versione dell'applicazione.

Dopo aver dato indicazioni sulle nuove funzionalità previste e su queste ultime siano state collocate all'interna dell'architettura del sistema è giunto il momento di entrare più nel dettaglio di come il tutto sia stato effettivamente implementato e trasportato dal puro ambito della progettazione a quello più concreto della programmazione, arrivando alla fine ad avere un prototipo funzionante in grado di mostrare come la strada intrapresa sia effettivamente in grado di fornire risultati utili allo scopo che ci si è prefissati.

Nel resto di questo capitolo verranno presentate le tecnologie scelte per la realizzazione dell'applicazione e come esse siano state utilizzate per rispettare i vincoli e requisiti formulati all'inizio, fornendo anche estratti di codice in modo da dare mostrare effettivamente il come si è proceduto.

### *4.1. Tecnologie scelte*

Vediamo ora di mostrare quali tecnologie si è deciso di utilizzare e le motivazioni legate a tali scelte.

Un'applicazione come Virtuose per poter tener fede ai propositi espressi durante la sua progettazione va suddivisa in diverse parti logiche, ciascuna eventualmente composta da più elementi fisicamente distinti, e per ognuna di esse si deve trovare il metodo implementativo ritenuto migliore per la sua realizzazione, eventualmente scendendo a compromessi ove necessario; nel nostro caso i diversi ambiti su cui porre l'attenzione per individuare le corrette tecnologie da utilizzare sono stati individuati in database e scambio dati, logica applicativa e costruzione interfaccia utente.

Per quanto riguarda la parte di database si è deciso di continuare a lavorare su un modello basato su XML, basandosi sul lavoro svolto nel precedente lavoro di tesi che aveva portato alla realizzazione della seconda versione del database di Virtuose, in quanto tale tecnologia si rivela essere in continuo miglioramento sia per quanto riguarda gli applicativi studiati per la gestione di dati memorizzati in tale formato sia per i metodi ad essa associati per l'organizzazione delle informazioni e la loro manipolazione. Oltre per questo continuo lavoro su XML e ciò ad esso collegato si è deciso di proseguire su tale strada dato che si è voluto puntare su un numero ristretto di concetti da usare nel sistema, cercando il più possibile di rendere il tutto uniforme dal punto di vista del trattamento di entità comuni presenti nelle varie aree di Virtuose ed infatti sfruttare un database basato su XML ha permesso di utilizzare la stessa tecnologia sia per i dati in esso memorizzati sia per i dati scambiati tra i vari moduli presenti.

Un'ulteriore motivazione legata all'uso dell'XML può essere riscontrata nel fatto che sfruttando tale linguaggio è possibile associare informazioni semantiche, grazie ad una scelta accurata dei nomi dei tag, degli attributi e dell'organizzazione di entrambi, ai dati

modellati attraverso di esso ed in questo modo si ha modo di dare maggiore chiarezza a quanto memorizzato nel database, rendendo ancor più semplice la gestione e comprensione dei vari messaggi e dei tipi cui essi fanno riferimento.

Per la parte di logica applicativa è stato deciso di utilizzare Java per i seguenti motivi:

- il paradigma ad oggetti su cui si basa tale linguaggio di programmazione è l'ideale per la realizzazione di applicazioni già progettate ed ideate per essere strutturate secondo un criterio modulare con suddivisione dei compiti tra diverse unità funzionali
- l'alta diffusione di Java nel mondo della programmazione ha portato ad avere grandi quantitativi di librerie liberamente disponibili da utilizzare per risolvere problemi comuni e da usare come fonti di codice dalle quali prendere spunto per i propri bisogni
- offre la possibilità di creare software altamente portabile su architetture di vario genere senza dover ricorrere ad interventi di grande impatto sul codice già realizzato.
- Permette di continuare nel proposito di sfruttare solo tecnologie ormai considerate standard de facto nell'ambito della realizzazione di applicazioni, soprattutto per il modo web

Per quanto riguarda la creazione dell'interfaccia utente è stato deciso di utilizzare dei metodi tali da rendere completamente disaccoppiati i dati dalla loro rappresentazione, fornendo al contempo un sistema in grado di essere facilmente implementabile e altamente flessibile. Per poter raggiungere questi obiettivi ed al contempo quello di utilizzare tecnologie standard si è giunti alla conclusione di suddividere la fase di gestione grafica in due, una parte dedicata all'aspetto generale e una dedicata ai contenuti specifici.

Per la presentazione dei contenuti contenuti nel sistema è stato deciso di sfruttare la tecnologia XSLT che permette di riorganizzare, anche in modo radicale, un blocco di

informazioni codificate in XML. L'aspetto molto interessante di questa soluzione è la possibilità di agire sui dati grezzi, completamente privi di informazioni e formattazioni stilistico-grafiche, in modo molto mirato e totalmente personalizzabile, con pochissimi vincoli e restrizioni (volendo è possibile avere al termine della trasformazione anche qualcosa di assolutamente diverso nella struttura rispetto al dato iniziale).

Per la realizzazione dell'interfaccia utente generale, facente da contorno ai contenuti e alla loro rappresentazione, si è deciso di basarsi su Jetspeed 2 [AJ2], un framework sviluppato dalla Apache Foundation per la realizzazione di portali e applicazioni basate sull'uso di Portlet [JPS], nel nostro caso usate come strutture di impaginazione per le parti grafiche create a partire dai contenuti, e con un proprio linguaggio per creazione di modelli grafici per la costruzione vera e propria dell'aspetto generale relativo all'interfaccia da presentare.

### *4.2. Scelta Database XML*

Dopo aver fissato la tecnologia da usare per la memorizzazione e organizzazione dei dati si è passati alla scelta di quale DBMS sfruttare, ponendo come requisiti il supporto ad XML, la possibilità di utilizzare strumenti standard per la manipolazione di quest'ultimo e facilità di integrazione con Java.

Come primo passo è stata effettuata una ricerca dei software disponibili con queste caratteristiche e su di essi si è provveduto a fare una prima scrematura in base a criteri di alto livello; terminata questa fase si è giunti ad avere tre possibili candidati sui quali è stata effettuata un'analisi più approfondita in modo da selezionare il migliore per i nostri scopi.

Vediamo quali sono stati i DBMS analizzati a fondo e quali considerazioni sono emerse su di essi.

- ***Ozone***

Ozone [OZO] è un software open source scritto in Java la cui struttura prevede un processo server, sul quale è scaricata tutta la logica per agire fisicamente sui dati, e un insieme di librerie, compatibili con le API XML:DB [XDB], realizzate per fornire le funzionalità necessarie al dialogo con tale componente.

La gestione dei dati XML avviene mediante l'uso di una propria implementazione del modello DOM [SHE02], denominata MonsterDOM, sulla quale operare mediante la DOM API di Java [JDOM] e in cui ogni nodo è memorizzato separatamente dagli altri, mantenendo comunque le informazioni gerarchiche e di relazione, in modo da rendere meno oneroso e più rapido il processo di accesso ai singoli elementi. L'uso di un derivato della tecnologia DOM permette di avere rapido accesso informazioni ma al contempo genera un maggior uso di risorse per le funzionalità di manipolazione dell'XML in quanto viene caricato in memoria l'intero insieme delle informazioni relative all'entità sulla quale si sta lavorando.

Per quanto riguarda le possibili tecniche di accesso ai dati presenti nel database, Ozone permette l'uso del linguaggio XPath [SIM02] per quanto riguarda la ricerca e l'individuazione delle informazioni mentre per l'aggiunta, la modifica e l'eliminazione dei nodi supporta l'utilizzo di XUpdate [XUPD].

- ***eXist***

eXist è un DBMS open source scritto interamente in Java, continuamente in sviluppo, e dichiaratamente orientato alla realizzazione di applicazioni web, anche se può essere utilizzato per altri ambiti senza incorrere in limitazioni delle funzionalità.

Il suo funzionamento prevede l'esecuzione di un processo server, generalmente integrato in una applicazione web gestita da un webserver minimale configurato automaticamente in fase di installazione, al quale connettersi tramite le funzioni messe a disposizione dalla API standard XML:DB per ottenere accesso ai dati.

La memorizzazione di questi ultimi avviene secondo un'organizzazione che prevede collezioni, entità il cui scopo è quello di fungere da contenitori in modo del tutto analogo a quello delle directory in un filesystem, e documenti, veri e propri portatori dei dati ed idealmente paragonabili a files; su questa struttura viene poi inserito un livello di ottimizzazione basato su indici in grado di fornire un aumento considerevole delle prestazioni, fatto reso ancor più evidente dall'uso di B-alberi e file paginati, e un livello di disaster recovery incentrato su transazioni in grado di fornire uno strumento per ripristinare il sistema in caso di errori.

Per quanto riguarda la modalità di parsing dell'XML memorizzato eXist sfrutta in maniera intensiva l'approccio SAX [SHE02] che, attraverso le relative librerie, permette di avere in memoria solo una parte del documento in analisi, trattando quest'ultimo come un flusso di dati sequenziale sul quale interagire tramite azioni legate al verificarsi di precisi eventi (apertura di un tag, chiusura di un tag, ...).

Chiudiamo il discorso su questo software e sulle sue caratteristiche dicendo che come tecnologie per l'individuazione,estrazione e modifica dei dati sono utilizzabili XPath, Xquery [BRU04] e Xupdate.

Da segnalare come la possibilità di sfruttare XQuery ampli enormemente le potenzialità d'azione sulle informazioni in quanto rende disponibile un vero e proprio linguaggio di programmazione orientato alla manipolazione dell' XML e in grado di ridurre la complessità della logica applicativa esterna al database scaricandone una parte su esso stesso.

- ***Berkeley DB XML***

Questo database per la memorizzazione di dati in formato XML si distingue dai due precedentemente illustrati in quanto non è un software, per così dire, autonomo visto che per gestire i dati veri e propri si basa su un altro prodotto integrato in esso, Berkeley DB [BDB], nel quale i documenti XML possono essere memorizzati sia come blocco unico sia come nodi informativi separati (nel caso si volessero ottimizzare le operazioni su piccole porzioni di dati). L'architettura sulla quale è basato Berkeley DB XML [BXML] è quindi divisa in due, una parte di più basso livello non orientata direttamente all'XML e che fornisce le funzionalità di memorizzazione, integrità dei dati tramite transazioni, gestione di accessi concorrenti, ottimizzazioni per migliorare le performance e quant'altro di utile per mantenere efficiente una base di dati, ed una posta a metà strada tra essa e l'applicazione avente come compito la gestione dei documenti XML, la loro indicizzazione e la fornitura del sistema di interrogazione e modifica delle informazioni.

Oltre a questa differenza di organizzazione interna è presente un ulteriore punto di distinzione, l'assenza della modalità client-server dato che tutta la logica applicativa di questo database è implementata all'interno della libreria che verrà usata dall'applicazione; questa caratteristica si ripercuote anche sull'insieme di funzioni usate per la manipolazione dei dati in quanto non può essere usato lo standard fornito da DB:XML e al suo posto si deve ricorrere forzatamente a quello specifico di B. DB XML.

Per quanto riguarda le possibilità di intervento per la modifica dei dati memorizzati, in questo caso, non ci si può basare su Xupdate ma si deve far ricorso a quanto fornito dalle librerie specificatamente sviluppate per questo software. Questo non avviene per le operazioni di reperimento dei dati in quando sono utilizzabili sia XPath che XQuery.

Di questi tre candidati Ozone è stato scartato in quanto il solo uso di XPath per la ricerca ed estrazione dei dati è sembrato essere un fattore troppo limitativo e a questo va aggiunto che il suo sviluppo è praticamente stato arrestato, ciò ci avrebbe portato ad una situazione simile a quanto accaduto in precedenza con Xindice, altro prodotto che si era rivelato piuttosto interessante ma al momento mancante di ulteriore sviluppo e manutenzione.

Berkeley DB XML, seppur dotato di ottime caratteristiche prestazionali, funzionalità di indubbia utilità quali una gestione molto avanzata delle transazioni e il supporto al linguaggio XQuery, è stato scartato a causa della mancanza di un architettura in grado di disaccoppiare "l'utilizzatore" dalla logica del database, ciò è dovuto al fatto che la gestione completa del database è integrata nella libreria stessa destinata ad essere utilizzata e quindi interventi su quest'ultimo componente avrebbero potuto avere effetti diretti sull'intero sistema; oltre a questo aspetto per certi versi problematico, un altro fattore che ha portato a scartare questo prodotto è quello di avere poca possibilità d'uso di standard affermati quali XUpdate e le funzionalità previste da DB:XML, fatto in netto contrasto con la volontà espressa più volte di utilizzare solo tecnologie comuni e globalmente accettate.

Date queste motivazioni, infine, è stato scelto eXist che nel corso del tempo si è rivelato essere un progetto sempre vivo, con piani di sviluppo orientati ad apportare miglioramenti su tutti i fronti e con già da ora supporto per vari standard utilizzabili per realizzare un prodotto valido da punto di vista qualitativo e anche con alte potenzialità di personalizzazione.

### 4.2.1. Struttura XML delle principali entità del database

Dopo aver mostrato le caratteristiche basilari del software al quale delegare la gestione "fisica" dei dati vediamo di dare qualche informazione sul come sono realmente organizzate a livello di codice XML le strutture delle principali entità pensate per essere utilizzate come fondamenta del sistema e legate in maniera diretta a quelli che sono stati presentati in precedenza come i moduli fondamentali.

- ***Conference***

Questa entità, legata al modulo omonimo, è quella destinata a portare informazioni circa le conferenze e a contenere i messaggi ad essa associati.

```
1.  <conference
2.      allowedMessagesType="..."
3.      cid="..."
4.      system="..."
5.      pid="..."
6.      aid="..."
7.      gid="..."
8.      group="..."     >
9.     <name>...</name>
10.    <hook>...</hook>
11.    <messages>
12.        ...
13.     </messages>
14. </conference>
```

Il tag *conference* contiene tutte le informazioni relative ad una data conferenza, i suoi attributi (linee 2-8) permettono di specificare i tipi di messaggio permessi (*allowedMessagesType* contiene la lista degli identificativi dei tipi di messaggio), l'identificativo della conferenza (*cid*), il dato relativo all'appartenenza delle conferenza stessa all'insieme di quelle fondamentali (*system*, valore booleano), l'identificazione della conferenza da cui quella attuale discende (*pid*), il riferimento all'eventuale

conferenza di cui quella in esame è un segnaposto (*aid*), l'indicazione del gruppo di appartenenza (*gid*), se esiste, ed infine l'indicazione se l'attuale conferenza è l'elemento rappresentativo di un dato gruppo (*group,* valore booleano).

Dopo gli attributi abbiamo i tag discendenti di quello principale, questi danno modo di memorizzare il nome della conferenza (linea 9, *name*), eventuali informazioni aggiuntive non fondamentali (linea 10, *hook*) e i messaggi interni alla conferenza (contenuti all'interno del tag *messages* nei suoi discendenti del tipo *message* presentato dopo questo, linea 11)

- ***Message***

In questa entità vengono memorizzate le informazioni e i dati contenuti all'interno dei messaggi posizionati all'interno delle conferenze.

```
1. <message
2.     mid="..."
3.     uid="..."
4.     pid="..."
5.     msgtype="..."    >
6.     <hook>...</hook>
7.     <content>...</content>
8. </message>
```

Anche in questo caso abbiamo un tag principale, *message*, che identifica un blocco informativo corrispondente al messaggio e i cui attributi forniscono i dati strutturali quali l'identificativo (linea 2, *mid*), l'utente associato al messaggio (linea 3, *uid*), il messaggio cui eventualmente quello in esame fa riferimento (linea 4, *pid*) ed il tipo di messaggio (linea 5, *msgtype*).

I dati non strettamente riferibili ad aspetti di sistema sono invece memorizzati in tag discendenti da quello principale, abbiamo un contenitore per informazioni non fondamentali (linea 6, *hook*) e quello che avrà al suo interno l'insieme dei dati realmente associati al messaggio in quanto portatore di contenuto (linea 7, *content*);

tale insieme è ovviamente strettamente legato alle specifiche di validazione contenute nel messaggio indicato dall'attributo *msgtype*.

Prima di dare spazio alle prossime entità bisogna dire che esse sono in realtà dei messaggi, dato che ne rispettano completamente la struttura, aventi un tipo associato prefissato e ben definito nel sistema e nei quali sono contenute le informazioni necessarie al corretto funzionamento dell'applicazione. Detto questo vedremo di fornire solo il contenuto del tag *content* presentato poco fa in quanto il resto delle informazioni è perfettamente conforme a quanto già detto.

- ***Profile***

La gestione dei dati per i profili è implementata all'interno di messaggi il cui contenuto segue questo schema:

```
1.  <profile>
2.      <name>...</name>
3.       <description>...</description>
4.       <views
5.          vids=”...” >
6.       </views>
7. </profile>
```

Il tag generale *profile* contiene quelli relativi alle informazioni vere e proprie quali il nome del profilo (linea 2, *name*), una descrizione (linea 3, *description*) e l'elenco degli identificati delle viste (attributo *vids* del tag *views*, linee 4-5) associate al profilo in questione.

- ***User***

La gestione dei dati per i profili è implementata all'interno di messaggi il cui contenuto segue questo schema:

```
1. <user
2.     uid="..."  >
3.     <authentication>
4.         <login>...</login>
5.         <password>...</password>
6.         <configuration>...</configuration>
7.     </authentication>
8.     <personal_data>
9.           <email>...</email>
10.          ...
11.     </personal_data>
12.    <profiles  pids="..."  />
13.    <moderatedConferences  cids="..."    />
14.</user>
```

Come si può notare la quantità di informazioni fondamentali associate ad un utente non è molto elevata, questo è dovuto al fatto che il possibile uso di diversi plugin per la connessione a basi di dati esterne per avere accesso a bacini d'utenza più ampia sposta verso essi parte delle informazioni; nel caso di un utente completamente gestito "in locale" tutti i dati personali saranno comunque memorizzabili in questa struttura ampliando la sezione *personal_data*.

L'insieme di tali informazioni è contenuto all'interno del tag *user*, in cui trova posto anche l'identificativo dell'utente (attributo *uid*), e suddiviso in diverse sezioni corrispondenti ad altrettanti tag: *authentication* (linea 3-7) nel quale abbiamo un identificativo alfanumerico per l'utente (linea 4, *login*), la password ad esso associata (linea 5, *password*), nel caso di utente da autenticare su un sistema esterno questo dato avrà un valore fasullo prestabilito ,non utilizzabile per effettuare l'accesso al

sistema e per risalire a quello reale, e un contenitore per eventuali regole aggiuntive utili alla gestione del processo di autenticazione (linea 6, *configuration*); la sezione successiva (linea 8-11, tag *personal_*data) è dedicata alle informazioni personali ed in particolar modo a quella obbligatoria riferita all'indirizzo email (linea 9,*email*); successivamente a questa abbiamo il tag *profiles* necessario per indicare a quali profili possono essere associati all'utente in questione, ciò è fatto indicando gli identificativi di tali messaggi particolari come valori dell'attributo *pids*; per concludere abbiamo il tag *moderatedConferences* che permette di fornire un elenco delle conferenze, tramite il loro identificativo univoco inserito nella lista interna all'attributo *cids*, per le quali l'utente può agire da moderatore.

- ***View***

Come già emerso in precedenza i permessi associati a conferenze e messaggi e gli stili grafici ad essi associati sono memorizzati nelle viste, la cui struttura piuttosto complessa è quella che segue.

```
1. <view>
2.     <name>...</name>
3.     <description>...</description>
4.     <conferences>
5.             <conference
6.                     cid=”...”
7.                      layout=”...”
8.                      messages_view=”...”    >
9.                      <messages>
10.                             <message msgtype=”...” >
11.                                     <perms
12.                                        view=”...”
13.                                        read=”...”
14.                                        write=”...”
15.                                        delete=”...”
```

```
16.                                    modify="..."
17.                                    superdelete="..."
18.                                    supermodify="..."  />
19.                                <style
20.                                    preview="..."
21.                                    view="..."
22.                                    message_view="..." />
23.                        </message>
24.                            ...
25.                    </messages>
26.                      <subconfs>
27.                            <conference
28.                                    cid="..."
29.                                    preview_style="..." />
30.                            ...
31.                     </subconfs>
32.             </conference>
33.             ...
34.       </conferences>
35. </view>
```

Come si può notare i dati relativi ad una vista sono divisi in una parte descrittiva composta dal tag *name* (linea 2) utile per associarle un nome mnemonico e dal tag *description* (linea 3) per associare una eventuale breve descrizione e da un'altra in cui sono elencate le conferenze, con i rispettivi tipi di messaggio, su cui la vista agisce.

Per ognuna delle conferenze elencate (tag conference, linea 5) abbiamo la presenza degli attributi: *cid* (linea 6) che indica l'identificativo della conferenza in questione, *layout* (linea 7) che indica il template globale associato alla visione di tale conferenza e *messages_view* (linea 8) che indica la modalità di visione dei messaggi presenti nella conferenza (thread, nested, ...); dopo questi attributi base troviamo due sezioni complesse, una specializzata nei messaggi appartenenti alla conferenza e l'altra dedicata alle eventuali sotto conferenze da visualizzare come discendenti di quella in esame.

Vediamo di analizzare come sono strutturate queste due parti del contenuto della vista e a cosa servono partendo da quella dedicata alle sottoconferenze (tag *subconfs*, linea 26); in questo caso ogni elemento presente identifica una data conferenza (tag *conference*, linea 27), che può essere visualizzata come discendente di quella su cui la vista sta dando informazioni, ed infatti abbiamo la presenza dell'identificatore univoco *cid* (attributo, linea 28) al quale si affianca *preview_style* (attributo, linea 29) che altro non è se non l'indicazione su quale template grafico utilizzare per dare un'anteprima di questa conferenza.

Ora vediamo come è strutturata la parte della vista che riguarda le indicazioni circa permessi e modalità di visualizzazione su un determinato tipo di messaggio presente nella conferenza. Per prima cosa notiamo la presenza dell'attributo *msgtype* (linea 10) che contiene l'identificativo del tipo di messaggio sul quale si stanno fornendo informazioni, poi abbiamo il tag *perms* (linea 11) al cui interno sono date tramite attributi, usando i valori booleani true e false, indicazioni sulle azioni permesse: *view* (linea 12) indica la visione di un'anteprima del messaggio, *read* (linea 13) il permesso di lettura, *write* (linea 14) la possibilità di inserire un messaggio di tale tipo, *delete* (linea 15) indica se è possibile rimuovere un proprio messaggio di tale tipo, *modify* (linea 16) specifica se si possono modificare i propri messaggi, *superdelete* (linea 17) se è possibile rimuovere uno qualsiasi dei messaggi di tale tipo indipendentemente dall'autore e per ultimo *supermodify* (linea 18) che si comporta come superdelete ma per quanto riguarda la possibilità di modifica.

Come ultima sezione che porta informazioni abbiamo il tag *style* (linea 19) che permette di specificare quale template utilizzare per l'anteprima del messaggio (attributo *preview*, linea 20) e quale per la lettura del messaggio (attributo *view,* linea 21), infine vengono fornite indicazioni su come deve essere visualizzato l'insieme dei messaggi avente quello in esame come capostipite (attributo *message_view*, linea 22).

### *4.3. Scambio dati in XML*

Come già accennato più volte il modo scelto per lo scambio di messaggi tra i moduli è basato su un protocollo che prevede l'invio di una richiesta sotto forma di XML aderente ad una struttura prefissata e di una risposta, anch'essa XML, non sottostante a regole preimposte ma solo dipendente dal modulo che la genera (per questo per ogni modulo è meglio definire durante la sua fase di progettazione come saranno strutturati i risultati delle sue elaborazioni da inviare al richiedente e rendere note tali informazioni al fine di agevolare il lavoro altrui di integrazione).

Vediamo ora di dare informazioni dettagliate su come deve essere organizzato un messaggio di richiesta inviato ad un modulo.

```
1. <virtuose>
2.     <request>
3.         <nome_modulo>
4.             <params>
5.                 <param>
6.                     <name>...</name>
7.                     <type>...</type>
8.                     <type>...</type>
9.                 </param>
10.                 . . .
11.             </params>
12.         </nome_modulo>
13.     </request>
14.</virtuose>
```

Una comunicazione inviata verso un modulo deve essere contenuta in un tag *virtuose,* al cui interno troviamo il tag *request* (linea 2) utilizzato per definire il tipo del messaggio inviato (al momento non ne sono previsti altri ma per dar modo di espandere in futuro il sistema di dialogo è stato deciso di introdurre questa ridondanza ora inutile) e che a sua volta vede sotto di se un altro tag, questa volta si tratta del nome del modulo con cui comunicare (linea 3); con questo termina l'insieme di dati

necessario affinché il nucleo applicativo riesca ad inoltrare la comunicazione al corretto destinatario.

La lista delle informazioni vere e proprie è contenuta sotto forma di elementi interni al tag *params* (linea 4) , ciascuno dei quali (*param*, linea 5) è caratterizzato da tre informazioni: il nome del parametro (linea 6, *name*), il tipo del parametro (linea 7, *type*) ed il "valore" del parametro (linea 8, *value*).

Tipo e valore sono informazioni strettamente legate in quanto la prima indica quale può essere il formato (*string, xml, int, float, base64; boolean non è presente in quanto si è deciso di sfruttare per esso i valori 0 e 1 del tipo interno*) della seconda e quindi come deve essere strutturato il suo contenuto.

### *4.4. Logica applicativa in Java*

La realizzazione della logica applicativa in Java ha permesso di strutturare il codice in modo del tutto corrispondente a quanto ottenuto al termine della progettazione astratta del sistema, infatti utilizzando il paradigma ad oggetti proprio del linguaggio scelto si è riusciti a modellare e collegare i vari componenti di Virtuose esattamente nel modo in cui erano stati pensati.

Il codice è stato raggruppato all'interno di un package (*it.virtuose*) a sua volta suddiviso in varie sottosezioni, ciascuna destinata a contenere elementi specializzati in un dato compito. Vediamo nel dettaglio questa organizzazione (dando anche alcune informazioni aggiuntive circa eventuali metodi di maggior interesse presenti in alcune classi ed interfacce).

#### *it.virtuose*

Questo package è quello principale, al suo interno troviamo due classi alle quali si aggiungono tutti i package da esso discendenti, vediamo prima le due classi e poi analizziamo il resto.

- Config: classe che implementa il sistema per la lettura e parsing del file di configurazione globale e che permette al resto del sistema di accedere a tali informazioni

  Metodi interessanti:

  - getModuleRemoteChannel: dato il nome di un modulo permette di risalire all'eventuale protocollo di comunicazione remoto attraverso il quale contattarlo
  - getInitParams: dato il nome di un modulo estrae i dati di configurazione per la sua inizializzazione
  - getRealModuleName: data una stringa per identificare un modulo restituisce il

nome reale del modulo cui essa fa riferimento, permette quindi di avere degli alias definiti per i vari moduli

- Core: questa classe è quella associata al nucleo applicativo che, come visto in precedenza, permette il dialogo tra i moduli e l'accesso a tutte le funzionalità offerte dal sistema

  Metodi interessanti:

  - talk_to: avvia il processo per l'inoltro di un messaggio di richiesta ad un modulo e per la ricezione della risposta con successivo invio della stessa al chiamante
  - getConfig: restituisce un riferimento al gestore della configurazione
  - loadExecModule: carica un modulo locale su comando del protocollo di comunicazione ed esegue le operazioni necessarie al completamento delle funzionalità richieste
  - loadExecModuleWithCache: analogo al metodo precedente ma impone l'uso della cache per il caricamento del modulo
  - loadExecModuleWithoutCache: analogo al metodo *loadExecModule* ma non permette l'uso del sistema di caching

***it.virtuose.communication***

Questo package serve per contenere tutte le classi ed interfacce utilizzate per le comunicazioni interne al sistema, in esso troviamo:

- Communication_Protocol: classe che fornisce le funzionalità necessarie al dialogo tra moduli sfruttando il protocollo di comunicazione base

  Metodo di interesse principale:

  - talk: implementa la chiamata ad un dato modulo, sia esso locale o remoto.
- Remote_Communication_Protocol: interfaccia che fornisce le specifiche che le classi

destinate ad implementare protocolli di comunicazione secondari (spesso per effettuare connessioni attraverso la rete) devono rispettare

Metodi fondamentali:

- init: necessario per l'inizializzazione dello specifico protocollo di comunicazione
- remote_talk: punto d'accesso al sistema di comunicazione tra un modulo o frammento di codice locale ed uno remoto

- InterPortletCommunication: classe che fornisce le funzionalità necessarie per il dialogo tra Portlet (per dettagli si veda il paragrafo *Sistema di comunicazione tra Portlet*)
- HTTP_Communication_Protocol: classe che implementa un protocollo di comunicazione remoto basato su HTTP, particolarità di questa classe è quella di essere strutturata in modo da essere utilizzabile anche come Servlet per la ricezione della chiamata sull'host remoto che ospita il modulo da contattare

  Metodi di interesse:

  - doGet e doPost: ricevono una richiesta HTTP contenente un messaggio da recapitare ad un dato modulo, anch'esso specificato nella richiesta stessa, e dopo aver inizializzato un ambiente conforme alle specifiche di Virtuose effettuano la chiamata al modulo in locale, restituendo poi il risultato di tale esecuzione sempre tramite il canale aperto per la ricezione dei dati
  - remote_talk: consente l'invio di una richiesta ad un modulo remoto attraverso l'uso del protocollo HTTP, ovviamente da modo anche di ricevere la risposta.

***it.virtuose.modules***

Questo package contiene tutto ciò che riguarda direttamente i moduli del sistema, in esso troviamo:

- Module: interfaccia che fornisce la struttura base, composta dai metodi fondamentali, che ogni modulo deve implementare

  Metodi:

  - init: permette di inizializzare il modulo in base ai suoi parametri di configurazione
  - close: disattiva il modulo dopo che è stato richiesto il suo intervento
  - exec: riceve il messaggio XML da interpretare per ricavare quale funzionalità è stata chiesta e, dopo aver eseguito i compiti ad essa connessi, restituisce il risultato dell'elaborazione

- GenericModule: classe di esempio per un modulo generico
- GenericUsers: classe per la definizione del modulo di gestione utenti.

  Metodi di particolare interesse:

  - init: inizializzazione del modulo generico e caricamento dei vari plugin utilizzabili per l'autenticazione e gestione degli utenti
  - getPlugin: dato un utente permette di ottenere un riferimento al plugin da utilizzare per le operazioni sui suoi dati
  - exec: punto di transito di tutte le richieste provenienti da altri moduli o frammenti di codice; è suo compito decidere quale plugin utilizzare per operare su un dato utente ed eventualmente eseguire una sincronizzazione tra il database utenti locale e quello esterno cui si è fatto accesso.

- Messages: classe relativa al modulo per la gestione dei messaggi (i metodi di questa classe sono quelli necessari a soddisfare la dipendenza da *Module* e a garantire le

funzionalità elencate nel paragrafo *Principali moduli e relative Funzionalità*)

- Conferences: classe relativa al modulo per la gestione delle conferenze (i metodi di questa classe sono quelli necessari a soddisfare la dipendenza da *Module* e a garantire le funzionalità elencate nel paragrafo *Principali moduli e relative Funzionalità*)
- Views: classe relativa al modulo per la gestione delle viste (i metodi di questa classe sono quelli necessari a soddisfare la dipendenza da *Module* e a garantire le funzionalità elencate nel paragrafo *Principali moduli e relative Funzionalità*)
- Profiles: classe relativa al modulo per la gestione dei profili (i metodi di questa classe sono quelli necessari a soddisfare la dipendenza da *Module* e a garantire le funzionalità elencate nel paragrafo *Principali moduli e relative Funzionalità*)
- StorageExistDb: classe che fornisce le funzionalità de gestione del database XML sfruttando quanto messo a disposizione da eXist.

  Al suo interno abbiamo come metodi principali:

  - new_id: dato un "prefisso" restituisce un indice univoco basato su esso
  - init: effettua il collegamento al database da usare poi per l'inoltro delle query
  - exec: esegue sempre il solito compito di ricevere una richiesta, eseguire i compiti ad essa associati e restituire il risultato
  - do_query: determina il tipo di query da eseguire e richiama il metodo ad esso associato
  - do_get_query: esegue una query di selezione basata su XPath o Xquery
  - do_modify_query: esegue una query di modifica dei dati basata su XUpdate

**it.virtuose.modules.storage_plugins**

Questo è il package al cui interno trovano posto i plugin utilizzati dal modulo di gestione

del database per ottenere le specifiche funzionalità necessarie ad interagire con i dati memorizzati, vediamo i particolari dei metodi relativi all'interfaccia *StoragePlugin* sulla quale devono essere basati tutti i plugin relativi al database*:*

- createStorageWorker: a partire da un insieme di dati e dall'azione da eseguire deve restituire un oggetto che possa essere utilizzato dal modulo di gestione del database per assolvere ai propri compiti

Nel nostro caso particolare, in cui si sfrutta eXist per operare sui dati, quanto restituito dal metodo *createStorageWorker* di uno qualsiasi dei plugin associati al modulo di gestione del database altro non è se non una stringa rappresentante la query da eseguire ed il relativo tipo.

**it.virtuose.modules.user_plugins**

Questo package è utilizzato per contenere i vari plugin necessari al modulo di gestione utenti per interagire con le varie fonti di dati. Al suo interno è presente l'interfaccia *UserPlugin* sulla quale devono essere basati tutti i plugin di gestione utenti, vediamone i dettagli a livello di metodi:

- init: solito metodo per l'inizializzazione del plugin in base a determinati dati di configurazione
- exec: permette l'esecuzione delle operazioni legate ad una data funzionalità richiesta
- synchronize:metodo che permette di sincronizzare il database utenti locale con quello esterno su cui si è operato tramite il plugin
- allowed_actions: restituisce l'elenco delle operazioni sugli utenti effettuabili sfruttando il plugin
- is_user_valid: verifica se un dato utente è gestibile attraverso il plugin in esame

***it.virtuose.utils***

Questo package ha come scopo il contenere tutte quelle parti di codice che possono

essere utilizzate come librerie dal resto del sistema. Al suo interno troviamo vari altri package ed una classe:

- IndexHandler: classe che implementa un sistema per operare su degli indici univoci (utilizzata molto dal database per la creazione degli identificativi associati alle varie entità)

***it.virtuose.utils.modules***

Package contenente varie utility da utilizzare per avere accesso a delle funzionalità relative ai moduli ma non direttamente implementate in essi e altre utilizzate dai moduli stessi, tra esse citiamo:

- GetStandardRequestType: classe per poter leggere da un messaggio il tipo di richiesta ricevuta, usata principalmente in una forma estesa dai vari moduli
- RequestBuilder: classe per la creazione e manipolazione di un messaggio di richiesta da inviare.

  Metodi interessanti:

  - addModule: inserisce nella richiesta il nome del modulo da contattare
  - addParam: inserisce nella richiesta un parametro (nome, tipo e valore)
  - addParamString, addParamInt, addParamXml, addParamFloat, addParamBase64: inseriscono un parametro del tipo indicato nel nome del metodo
  - removeParam: elimina il parametro indicato dai dati memorizzati per la richiesta
  - reset: elimina tutte le informazioni inserite per la creazione di una richiesta
  - create: dati tutte le informazioni inserite relative alla richiesta restituisce il documento XML da inviare al modulo destinatario

***it.virtuose.utils.xml***

Package il cui scopo è contenere funzionalità per la gestione e manipolazione dell'XML., tra esse abbiamo:

- SAX_EH: classe per gestire eventi del parser SAX
- XMLUtils: classe che fornisce strumenti di manipolazione dell'XML sia in formato di stringa di testo che come oggetto DOM
- XSLTransformer: classe che permette di operare trasformazioni di un documento XML dato uno XSLT
- XSDValidator: classe che permette di validare un frammento di codice XML seguendo le regole definite in un documento XML-Schema
- XQueryBuilder: classe che permette di creare una query secondo la sintassi XQuery per il database XML

***it.virtuose.webcommunity***

Questo package è quello espressamente dedicato al codice specifico per la realizzazione del sistema orientato alle comunità virtuali. In esso troviamo:

- NavigationPage: classe che permette la gestione dei dati (data e ora di accesso e parametri della richiesta) relativi ad una pagina della community visualizzata dall'utente.
- NavigationHistory: classe che permette di gestire la cronologia delle pagine visitate da un utente durante il suo periodo di navigazione (sfrutta la classe precedente per la gestione dei dati sulle singole pagine).

  Metodi principali:

  - getPage: dato un indice restituisce la pagina (NavigationPage) ad esso associata
  - getBack: restituisce la pagina visualizzata prima di quella attiva al momento della chiamata

- addPage: aggiunge una nuova pagina alla cronologia di navigazione
- removePage: dato un indice rimuove i dati della pagina ad esso associata

***it.virtuose.webcommunity.layouts***

Package al cui interno sono contenute le funzionalità utili alla creazione dei layout grafici:

- LayoutGenerator: interfaccia che fornisce le specifiche che una classe per la gestione dei layout deve rispettare.

  Metodi:

  - load: permette di caricare il layout indicato e, dopo aver eventualmente eseguito le operazioni indicate nelle opzioni passate come parametro, di restituirlo al richiedente
  - parse: dato un layout ed un documento XML permette di eseguire le trasformazioni del caso su quest'ultimo, eventualmente sfruttando le indicazioni aggiuntive indicate nelle opzioni ricevute tra i parametri
  - create: permette di invocare *load* e *create* in una sola chiamata
  - setBaseDir: permette di impostare la directory nella quale cercare i layout da utilizzare
- GenericLayout: classe che implementa LayoutGenerator ma che in realtà non fornisce funzionalità utilizzabili nella praticamente
- HtmlLayout: classe che estende GenericLayout e che fornisce tutto il necessario per la creazione di codice HTML da utilizzare nella realizzazione dell'interfaccia utente a partire da dati in XML e regole per la trasformazione XSL.

***it.virtuose.webcommunity.portlets***

Questo è il package al cui interno sono raggruppate tutte le portlet realizzate per fornire

gli elementi funzionali delle pagine web dell'applicazione di community (maggiori dettagli possono essere trovati nel paragrafo *Portlet*):

- VirtuosePortlet: questa classe implementa una Portlet e tutti le funzionalità che essa deve avere per sfruttare l'architettura di Virtuose vista fino ad ora, ogni portlet realizzata per il gestore di comunità virtuali deve estendere questa in modo da avere pieno accesso ai vari sistemi di comunicazione per il dialogo con i moduli, con il gestore della configurazione e con le altre portlet.
- StartupPageSettings: questa classe fornisce tutta una serie di funzionalità necessarie ad inizializzare lo stato del sistema, ad eseguire controlli sui permessi e a definire eventuali azioni particolari che il resto delle portlet presenti nella pagina dovranno seguire
- ClosePage: il compito di questa classe è fornire una portlet in grado di terminare tutti le connessioni,i processi rimasti attivi e le varie operazioni che alla fine della creazione di una pagina per l'interazione con la community devono essere deinizializzate.
- Menu: la creazione del menu dell'utente per la navigazione e l'interazione con il sistema è a carico della portlet implementata da questa classe
- ViewBody: questa classe fornisce l'implementazione di una portlet in grado di analizzare i dati relativi all'azione richiesta dall'utente e alle elaborazioni eseguite da Menu e StartupPageSettings ed in base a questi decidere quale sia la portlet specifica da richiamare per portare a compimento quanto dovuto.
- ViewConferenceBody: attraverso questa classe viene realizzata la portlet destinata a visualizzare una data conferenza richiesta dall'utente secondo i criteri imposti nella vista corretta.
- ViewMessageBody: attraverso questa classe viene realizzata la portlet destinata a visualizzare un dato messaggio richiesto dall'utente, il tutto rispettando i criteri

presenti nella vista corretta.

- ChooseProfileBody: questa classe fornisce l'implementazione della portlet necessaria alla creazione del menu di scelta di quale dei propri profili utilizzare da parte dell'utente.

### *4.5. Realizzazione layout tramite XSL*

Durante la pianificazione e l'implementazione del prototipo di Virtuose 3.0 è stata, come già più volte affermato, seguita la linea guida di tenere ben separati i dati da quella che sarebbe stata la loro rappresentazione visiva da fornire all'utente e, dato che la gestione dei contenuti è stata realizzata sfruttando XML, è parso naturale sfruttare il linguaggio XSLT e i relativi documenti XSL per implementare la logica ti trasformazione da pura informazione ad informazione strutturata graficamente visto che tale tecnologia è nata proprio con lo scopo di fornire uno strumento per riorganizzare un frammento in XML portandolo in una nuova forma.

Questo processo di "modifica" opera principalmente sulla base di *template* definiti all'interno del file XSL che permettono di identificare, attraverso l'uso di regole basate su XPath, i nodi del documento XML su cui agire allo stesso tempo forniscono tutte le istruzioni necessarie a compiere la trasformazione.

Vediamo un esempio pratico basato su un frammento XML corrispondente ad un messaggio molto semplice, ma del tutto simile ad uno che potrebbe essere realmente presente nel database di Virtuose e da esso estratto e parzialmente elaborato prima di essere presentato all'utente, ed un file XSL preso dall'insieme di quelli generici preparati per lo sviluppo e test del prototipo dell'applicazione realizzato in questa tesi.

DATI XML:

```
1. <?xml version="1.0" ?>
2.  <message mid="m1" uid=”u1” pid=”m1” msgtype=”m10”>
3.          <hook />
4.          <content>
5.             <simple_message>
6.                 <title>New discussion</title>
7.                 <text>First Message!!!</text>
8.              </simple_message>
9.          </content>
10.         <replies>1</replies>
11. </message>
```

FILE XSL:

```
1. <?xml version="1.0" ?>
2. <xsl:stylesheet   version="1.0"
3.                     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  >
4. <xsl:variable name="msgtype"></xsl:variable>
5.
6.   <xsl:template match="//message[@msgtype='{$msgtype}']">
7.      <b><xsl:value-of select="//title" /></b>:
8.      <br/> <xsl:value-of select="//text" />
9.      <xsl:if test="./replies &gt; 0">
10.         <br/>
11.         <a
   href="portal/virtuose_jetspeed.psml?object=message&amp;action=read&amp;mess
   age_id={./@mid}">Leggi risposte</a>
12.     </xsl:if>
13.     <br/><br/>
14.  </xsl:template>
15.
16.</xsl:stylesheet>
```

Bisogna far notare prima di tutto che il sistema per la creazione dell'interfaccia utente prima di eseguire la trasformazione va a modificare il documento XSLT andando a dare un valore  a tutte quelle sue porzioni di codice che identificano una variabile e che non sono state valorizzate direttamente in fase di stesura (FILE XSL, linea 4), questo da modo di poter avere avere un foglio di stile XSLT generico che verrà specializzato a runtime ogni volta che sarò richiesto. A questo punto possiamo passare a dar spiegazioni in merito al procedimento messo in atto dal trasformatore e fornire infine il risultato.

Come detto il tutto si basa sull'identificazione dei vari nodi da elaborare e nel nostro caso si ha l'estrazione (FILE XSL, linea 6) del nodo corrispondente al tag *message* (senza indicazioni sulla sua "posizione gerarchica" nel documento) avente l'attributo *mid* con valore corrispondente alla variabile indicata ad inizio file. Una volta selezionato questo elemento vengono applicate le regole di trasformazione (FILE XSL, linea 7-13): riscrittura del titolo in grassetto (FILE XSL, linea 7), scrittura del corpo del messaggio in una nuova riga (FILE XSL, linea 8), controllo della presenza di eventuali repliche al messaggio in esame (FILE XSL, linea 9) ed in caso affermativo preparazione in una nuova riga del link da utilizzare per accedere a tale contenuto (FILE XSL, linea 10-11) ed infine imposizione di due righe vuote (FILE XSL, linea 13).

Come si sarà notato il file con le specifiche per la trasformazione da eseguire è stato studiato per creare un frammento di una pagina HTML dato che nel nostro caso lo scopo era quello di realizzare una parte dell'interfaccia web per la visione di un messaggio, nulla vieta comunque di trasformare l'XML in un altro formato.

Il risultato finale dell'applicazione del file XSL, eventualmente ed opportunamente modificato a runtime, è quello visibile nel seguente codice:

RISULTATO TRASFORMAZIONE:

```
1.      <b>New discussion</b>:
2.      <br/> First Message!!!
3.      <br/>
4.         <a
   href="portal/virtuose_jetspeed.psml?object=message&amp;action=read&amp;mess
   age_id=m1">Leggi risposte</a>
5.      <br/><br/>
```

### *4.6. Uso di Jetspeed*

Per la gestione dell'infrastruttura globale su cui basare la costruzione delle varie pagine dell'interfaccia web di Virtuose è stato utilizzato Jetspeed.

La peculiarità di questa soluzione è la possibilità di comporre le varie pagine web utilizzate come vera e propria interfaccia tra l'utente e l'applicazione indicando per esse una struttura generale nella quale inserire, secondo necessità, le varie portlet destinate a fornire i dettagli specifici del tipo di contenuto da presentare.

Vedremo ora come è stata integrata al suo interno l'applicazione realizzata in questa tesi, dando dettagli sul come sono state strutturate e realizzate le varie portlet necessarie a fornire gli elementi contenutistici e che sfruttano le metodologie di trasformazione dell'XML tramite XSLT viste in precedenza, il sistema per permettere lo scambio di dati tra queste ultime ed infine come è stata realizzata la parte di interfaccia utente non direttamente controllata dalle portlet.

#### **4.6.1. Integrazione con Jetspeed**

Jetspeed non pone particolari problemi per quanto riguarda l'integrare al suo interno un'applicazione, soprattutto se già pensata per essere inserita in un ambiente basato su portlet. Vediamo cosa è stato necessario fare per adattare il prototipo di Virtuose 3.0 all'ambiente di esecuzione scelto per esso.

Innanzitutto sono stati raggruppati tutti i file in formato *bytecode* relativi a classi e interfacce all'interno di una struttura di directory che rispecchiasse appieno l'organizzazione dei namespace realizzati (it/, it/virtuose, it/virtuose/modules/, ...) e poi tutto ciò è stato inserito all'interno di una directory *classes.*

Fatto questo tutti i file *jar* contenenti le librerie esterne a Virtuose ma necessarie al suo corretto funzionamento sono stati inseriti all'interno della directory *lib*.

Con queste due operazioni si è arrivati ad avere una suddivisione del codice in due blocchi, uno dedicato a tutto quello sviluppato nel corso di questa tesi e l'altro a tutto ciò di esterno che è stato necessario utilizzare, esattamente secondo i requisiti imposti da Jetspeed.

Altro requisito da soddisfare è il fornire informazioni circa la strutture delle pagine web, ciò è realizzabile creando una nuova directory, allo stesso livello delle due già presentate, denominata *pages* all'interno della quale inserire dei file XML con estensione *.psml [PSML];* ognuno di questi file avrà al suo interno le specifiche per una data pagina, compreso l'elenco delle portlet da visualizzare (i dettagli saranno forniti nel paragrafo *Template grafici di Jetspeed*).

Per i nostri scopi abbiamo realizzato una sola pagina *(virtuose_jetspeed.psml)* che permetterà comunque di accedere a tutta una serie di funzionalità attraverso meccanismi di esecuzione di codice differente in funzione dei parametri ricevuti.

Restano ancora due file necessari affinchè Jetspeed sia in grado di eseguire correttamente la nostra applicazione, una dedicato ad identificare la stessa ed un altro fondamentale per indicare dove trovare il codice relativo alle portlet e come identificare queste ultime.

Il primo di questi due è il file *web.xml* al cui interno trova posto una descrizione generica dell'applicazione ed il nome con la quale renderla accessibile mentre il secondo è il file *portlet.xml* nel quale per ogni portlet sviluppata e direttamente indicata nei file .psml deve essere presente una serie di dati tra i quali un identificativo univoco, l'indicazione della classe (compreso il package di appartenenza), i tipi di visualizzazione permessi (VIEW,EDIT,...) e il Mime-Type ad essi associato (text/html, text/xml, ...).

Tutti questi file e directory devono a loro volta essere inseriti all'interno di un contenitore generale, la directory *WEB-INF* alla quale verrà affiancata quella denominata *META-INF* nella quale saranno presenti eventuali altre indicazioni aggiuntive riferite.

Una panoramica di questa organizzazione è visibile in Figura 8.

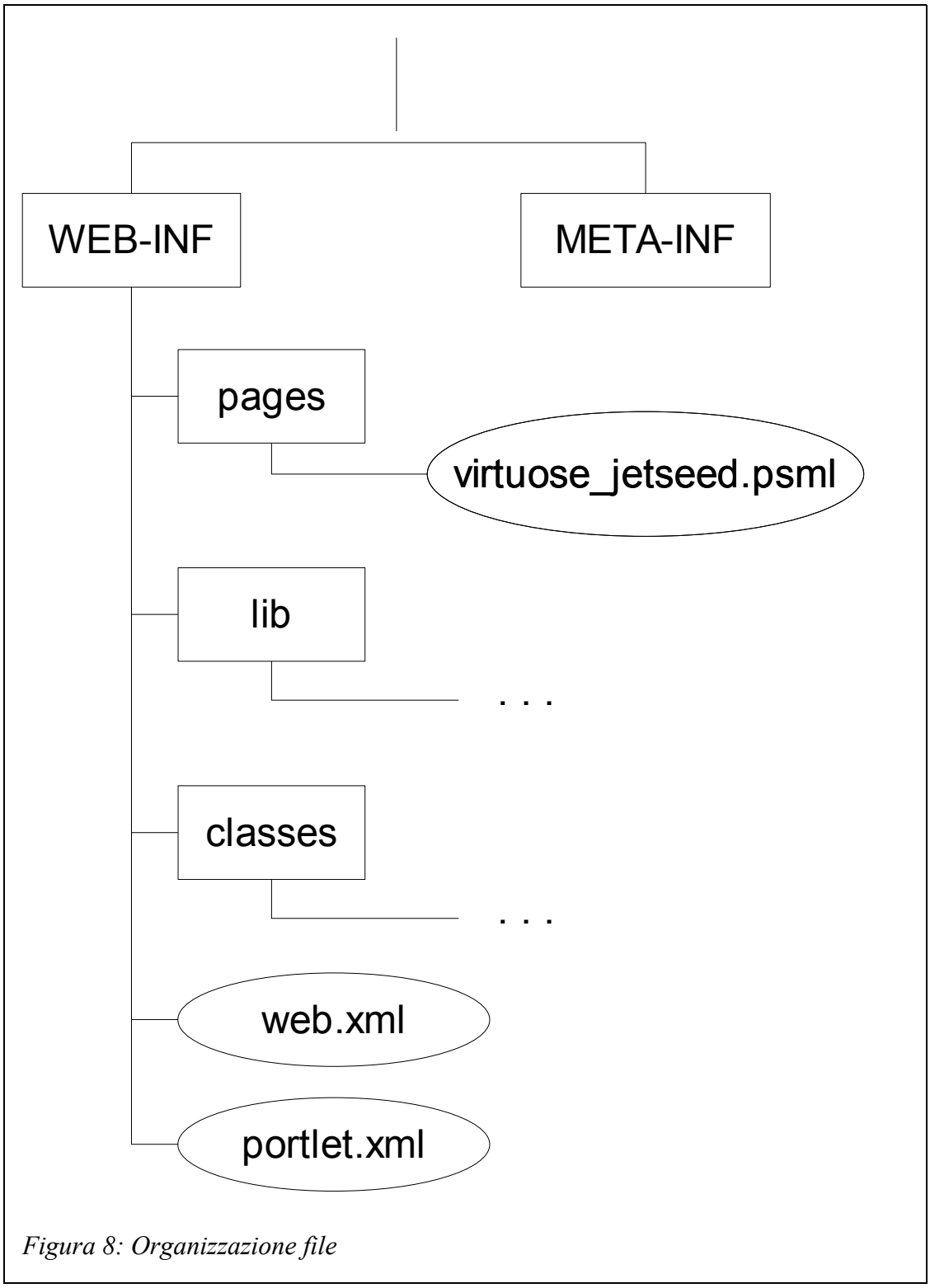


*Figura 8: Organizzazione file*

Tutto questo lavoro da solo però non permette di integrare subito Virtuose all'interno di Jetspeed, per farlo infatti occorre generare a partire da questo insieme di directory un file con estensione .*war*, che altro non è se non un file compresso in formato *ZIP* con al suo interno tutto quanto appena presentato. Una volta ottenuto ciò è sufficiente copiare quest'ultimo nella directory *deploy* presente all'interno dell'insieme di quelle di Jetspeed che si occuperà subito di decomprimere tale file e di posizionare nella corretta posizione tutti gli elementi presenti al suo interno.

Con ciò si vedrà realizzata l'integrazione, processo che da solo però non permette di avere un funzionamento corretto di Virtuose 3.0 dato che non si sono dati riferimenti a dove trovare il suo file di configurazione ma questa è un'operazione facilmente eseguibile e nel nostro caso si è scelto di creare un link simbolico, all'interno della directory dalla quale avviare l'intero ambiente di Jetspeed, facente riferimento al vero file *virtuose.conf.xml* (ovviamente è stato usato questo stesso nome).

### 4.6.2. Portlet

Data la predisposizione di Jetspeed alla realizzazione di pagine attraverso l'aggregazione di diverse portlet, eventualmente con l'aggiunta di altri elementi grafici generati dal suo sistema di templating, è stato necessario studiare quali funzionalità base una di esse dovesse avere per poter sfruttare appieno l'architettura di Virtuose; dato che quest'ultimo è basato principalmente su un sistema di dialogo tra moduli (o più in generale tra elementi di codice distinti) e che tale possibilità di comunicazione è resa possibile sfruttando un dato elemento fino ad ora denominato nucleo applicativo risulta evidente come ogni portlet realizzata per la nostra applicazione debba avere al suo interno un oggetto che implementi tale componente, oltre ovviamente a tutto il necessario per scambiare dati con le altre portlet presenti nella pagina (si veda il prossimo paragrafo per maggiori informazioni e dettagli su quest'ultimo aspetto).

Oltre a questi due aspetti legati principalmente al buon funzionamento della portlet e alla sua corretta integrazione all'interno di Virtuose ne è stato individuato un altro non irrilevante, quello relativo alla creazione dell'interfaccia utente a partire da dati in XML e basata su XSLT e gestito dalle opportune classi presenti nel package, presentato in precedenza, *it.virtuose.webcommunity.layouts*.

Tutte queste funzionalità per essere velocemente integrabili nelle varie portlet da utilizzare per i nostri scopi sono state implementate all'interno di una classe generica, presente anch'essa nell'ultimo package citato, denominata *VirtuosePortlet* e che è stata pensata per essere la base di partenza sulla quale costruire le altre, alle quali non resta che estenderla per poi aggiungere le proprie funzionalità specifiche a quelle ereditate. Vediamo quali sono i metodi principali presenti in essa e quindi ereditati e ridefinibili nelle altre:

- setLayout: crea l'oggetto necessario per la gestione del layout grafico relativo alla

portlet in base alle indicazioni ricevute come parametro

- renderLayout: dato un documento XML ed un documento XSLT, con l'aggiunte di eventuali opzioni, esegue la trasformazione e restituisce il risultato di tale operazione
- init_ipc: inizializza il sistema per permettere alla portlet di scambiare dati e sincronizzarsi con le sue simili presenti nella pagina
- is_ipc_initialized: permette di determinare se il sistema di comunicazione tra portlet è stato attivato o meno
- setSessionParameter: consente di impostare il valore per una data variabile di sessione, in caso di mancata inizializzazione del sistema di interscambio dati tra portlet genera un'eccezione
- getWaitingSessionParameter: permette di ottenere il valore di una variabile di sessione restando eventualmente in attesa fino a quando non viene creata. Nel caso il tempo massimo indicato si esaurisca o manchi il supporto per il dialogo tra porlet viene lanciata un'eccezione
- getSessionParameter: sfruttando il metodo appena visto permette di estrarre il valore di una variabile di sessione, nel caso ne abbia uno. Anche in questo caso viene generata un'eccezione nel caso non si possa accedere all'infrastruttura per lo scambio dati.
- sendMessageToModule: permette l'invio di un messaggio ad un dato modulo e la ricezione dell'eventuale risposta. Non richiede l'esistenza del destinatario della comunicazione, nel caso mancasse quest'ultimo verrà presupposta una risposta nulla.
- sendMessageToExistingModule: analogo al metodo precedente ma nel caso di mancata esistenza del modulo destinatario verrà generata un'eccezione.

Ovviamente oltre a questi metodi è presente anche quello, implementato come costruttore della classe specifica della portlet, per l'inizializzazione della struttura dati

relativa al nucleo applicativo specifico di Virtuose sul quale far affidamento per le comunicazioni con i moduli e per l'accesso ai dati di configurazione generali.

Per concludere il discorso sulle portlet vediamone qualcuna di quelle fondamentali di Virtuose 3.0 con dettagli sui metodi di maggior interesse presenti in esse:

- StartupPageSettings
  - doView: metodo specifico delle portlet per rispondere ad una richiesta di visualizzazione adattato nel nostro caso per permettere di inviare eventuali istruzioni per creare meccanismi di redirezione o modifica del flusso di navigazione
  - render: metodo specifico delle portlet per la preparazione dei contenuti da inviare al richiedente che per i nostri scopi è stato esteso in modo da eseguire i controlli di sicurezza di base e tutte quelle operazioni di inizializzazione delle variabili di controllo e sincronizzazione delle altre portlet presenti nella pagina
- Close
  - doView: metodo sfruttato per la distruzione di tutte le variabili di sessione utilizzate per guidare il processo di creazione della pagina a carico delle portlet
- Menu
  - doView: il metodo in questo caso è usato esattamente per assolvere i compiti per cui è stato ideato ed infatti permette di generare quella parte della pagina necessaria all'utente per svolgere compiti legati alla navigazione all'interno delle varie sezioni dell'applicazione
- ViewBody
  - doView: questo metodo è utilizzato per generare la porzione di contenuto principale da presentare all'utente basandosi sui risultati delle elaborazioni di *StartupPageSettings:* in caso si siano verificati errori verrà preparata una loro

spiegazione, in caso non ce ne siano verrà avviata la procedura per la generazione di quanto necessario in funzione dei parametri ricevuti ed eventualmente modificati dal sistema.

In base a queste ultime informazioni, infatti, questa portlet ricava quale delle altre presenti nel package richiamare e, dopo averne istanziata una copia, richiede ad essa l'esecuzione del metodo *doMyJob* che dovrà eseguire il vero e proprio lavoro di creazione dei contenuti da presentare all'utente.

### 4.6.3. Sistema di comunicazione tra Portlet

Il fatto di avere più portlet presenti in ciascuna pagina, ognuna di esse con una propria istanza dell'ambiente necessario per sfruttare le funzionalità di Virtuose, ha portato alla luce una problematica relativa alla comunicazione e alla sincronizzazione tra esse.

Jetspeed infatti per ognuna di esse crea un thread dedicato ed indipendente dagli altri ma non fornisce direttamente dei metodi per lo scambio di informazioni, situazione nel nostro caso aggravata dalla totale mancanza di informazioni a priori su quali portlet siano effettivamente in esecuzione e a che punto del loro ciclo di vita esse siano.

In questa situazione l'unico dato realmente condiviso e facilmente accessibile da ogni oggetto nella pagina è l'insieme delle variabili di sessione e per questo motivo è stato deciso di lavorare su di esso per la realizzazione dei meccanismi necessarie allo scambio di dati arrivando a definire in modo del tutto naturale, dato il tipo di funzionalità da realizzare e data la tipologia di oggetti su cui si è deciso di intervenire per l'implementazione, quali azioni si rivelano essere fondamentali: inizializzazione, lettura immediata, scrittura immediata e lettura con eventuale attesa delle variabili di sessione. Vediamo più nel dettaglio il loro funzionamento:

- scrittura immediata (metodo *set*): crea una nuova variabile di sessione avente per identificativo e valore quelli forniti (nel caso esista già una variabile con tale nome la modifica)
- lettura con attesa (metodo *get* con l'ausilio di quello *wait4*): cerca tra le variabili di sessione una corrispondente al nome ricevuto come parametro e se non la trova continua in questo processo fino a che non scade il quantitativo di tempo indicato dal chiamante; nel caso non si sia potuto ottenere un valore nonostante l'attesa viene lanciata un'eccezione.
- lettura immediata (metodo *get*): questa funzionalità è basata sulla precedente ed

infatti altro non è se non una chiamata ad essa avente il parametro indicante il tempo massimo di attesa con valore 0.

- inizializzazione (metodo *init*): permette di memorizzare il riferimento alla sessione su cui operare

Queste tre possibilità di azione relative alle comunicazioni tra portlet sono state implementate come metodi all'interno della classe *InterPortletCommunity* presentata in precedenza all'interno del package *it.virtuose.communication*, la quale a sua volta è utilizzata dalla classe *VirtuosePortlet* del package *it.virtuose.webcommunity.portlets* per la creazione e inizializzazione di un oggetto da fornire a tutte le portlet create estendendo essa stessa.

Ora dovrebbe essere chiaro il meccanismo attraverso il quale si è ottenuta la capacità di comunicazione, ma per quanto riguarda la sincronizzazione non si è detto nulla di esplicito, vediamo di chiarire anche questo punto.

Per poter dare ordine e controllare l'esecuzione di diverse unità applicative (nel nostro caso i thread cui sono associate le portlet) si deve avere la possibilità di interromperne temporaneamente il funzionamento, in modo da arrivare ad avere degli istanti in cui lo stato del sistema è in una conformazione piuttosto definita e senza rischio di generare effetti collaterali indesiderati. Attraverso la funzionalità di "lettura con attesa" è possibile realizzare questo sistema di controllo e sincronizzazione dal momento che sfruttando delle opportune variabili di sessione e i tentativi temporizzati di accesso alle stesse è possibile far si che una portlet resti in sospensione fino a quando verrà dato il segnale di proseguire.

Per far ciò ogni portlet dovrà creare ed inizializzare delle variabili di sessione, indicanti lo stato dall'esecuzione ed eventuali informazioni di interesse generale per l'applicazione, attraverso i cui valori verrà comunicato il verificarsi di dati eventi.

Un caso tipico è quello di far in modo di sospendere l'esecuzione di una portlet fino al

completamento di un'altra: questo si ottiene facendo si che quella interessata ad ottenere il sincronismo chieda al sistema di comunicazione tra portlet di accedere al valore della variabile indicante la terminazione dell'altra, indicando al contempo quanto tempo massimo restare in attesa di questo evento.

### 4.6.4. Template grafici di Jetspeed

Oltre a delegare la creazione degli aspetti visivi da presentare nelle pagine web generate dal codice integrato in esso, Jetspeed offre dei metodi sia per arricchire la struttura grafica generale sia per dare ordine a quanto prodotto internamente dalle portlet.

Tutte queste definizioni stilistiche sono fornite per ciascuna pagina accessibile dall'utente all'interno di file XML particolari con estensione PSML; vediamo il file usato  usiamolo come base per dare ulteriori spiegazioni.

```
1. <page id="virtuose_jetspeed_home">
2.   <defaults skin="orange"
3.               layout-decorator="tigris" portlet-decorator="gray-gradient" />
4.   <title>Virtuose</title>
5.
6.   <fragment id="pagina" type="layout" name="jetspeed-
   layouts::VelocityOneColumn">
7.     <fragment id="riga-1" type="portlet" name="virtuose_jetspeed::startup">
8.       <property layout="OneColumn" name="row" value="0" />
9.     </fragment>
10.      <fragment id="menu-1" type="portlet" name="virtuose_jetspeed::menu">
11.        <property layout="OneColumn" name="row" value="1" />
12.      </fragment>
13.      <fragment id="main-1" type="portlet" name="virtuose_jetspeed::main">
14.        <property layout="OneColumn" name="row" value="2" />
15.      </fragment>
16.    <fragment id="riga-3" type="portlet" name="virtuose_jetspeed::close">
17.      <property layout="OneColumn" name="row" value="3" />
18.    </fragment>
19.  </fragment>
20.
21.  <security-constraints>
22.    <security-constraints-ref>public-view</security-constraints-ref>
23.  </security-constraints>
24.
```

```
25.</page>
```

Per ogni pagina vengono indicati un identificativo univoco (linea 1, attributo *id*), un titolo descrittivo (linea 4, tag *title*) ed una serie di informazioni basilari circa il suo stile grafico memorizzate in due attributi appartenenti al tag *defaults*, *layout-decorator* (linea 3) e *portlet-decorator* (linea 3):

- layout-decorator permette di stabilire quale template utilizzare per generare le porzioni di contenuto corrispondenti alla testata e al piè di pagina, quali fogli di stile CSS [SCH04] usare e quali aspetti stilistici associare alle portlet in caso non se ne forniscano di specifici.

  Questi modelli grafici sono contenuti nella directory *decorations/layout/<nome_layout>* di Jetspeed e possono ovviamente essere modificati (o aggiunti di nuovi) a patto di rispettare la regola base che vuole la presenza dei file:

  - footer.vm: il piè di pagina, il suo contenuto può essere in formato JSP [BAK01] o Velocity Template Language [VTL].
  - header.vm: testata della pagina, anche per esso il contenuto può essere in formato JSP o Velocity Template Language .

    styles.css: file contenente le definizioni dei fogli di stile CSS da usare
  - decorator.properties: contenitore per informazioni circa il path in cui trovare gli altri file presentati e altri aspetti grafici
- portlet-decorator invece è necessario per indicare quale modello grafico utilizzare per la gestione degli aspetti stilistici relativi alla parte esterna delle portlet (ad esempio bordi, testi descrittivi, ...).

  I vari modelli predefiniti, o quelli aggiuntivi, corrispondenti a questa categoria sono

contenuti nella directory *decorations/portlet/<nome_layout>* di Jetspeed e sono caratterizzati dalla presenza obbligatoria di un solo file:

- decorator.properties: contenitore per tutte le informazioni necessarie al reperimento dei vari elementi costitutivi il template grafico in oggetto

Per quanto riguarda l'impaginazione Jetspeed prevede la possibilità di sfruttare tutta l'area visibile non ancora utilizzata suddividendola in sezione verticali (colonne), da un massimo di tre ad un minimo di una, nelle quali poi sarà possibile operare sfruttando ancora in modo ricorsivo questo tipo di organizzazione; questo porta a creare una griglia nella quale posizionare i contenuti da inserire nella pagina sfruttando un sistema di coordinate assimilabile a quello utilizzato per identificare dei punti in un piano cartesiano.

Tutte le modalità di suddivisione ed impaginazione sono implementate come particolari modelli grafici implementati sotto forma di portlet specifiche alle quali vengono affiancati dei file esterni di tipo Velocity Template Language o JSP.

Per il nostro prototipo è stato deciso di sfruttare una sola colonna nella quale mettere, dall'alto verso il basso, le portlet secondo l'ordine StartupPageSetting, Menu, ViewBody e Close.

Nel file in esame queste decisioni sono state tradotte creando una cella (tag *fragment*) principale (linea 6-19) destinata a definire una modalità di visualizzazione (linea 6, attributo *type* con valore *layout*) e assegnandole la tipologia VelocityOneColumn (linea 6, attributo *name*).

La disposizione verticale degli elementi è stata poi realizzata creando all'interno del contenitore appena indicato una serie di altre celle (linee 7-18) utilizzate per contenere le portlet (questa predisposizione è definita dal valore *portlet* dato all'attributo *layout* alle linee 7,10,13,16); ad esse, oltre ad un nome univoco (attributo *id*) ed un identificativo della portlet da visualizzare (attributo *name*), è stato associato anche un riferimento a quale posizione (riga) occupare in riferimento al layout scelto (tag *property*, linea

8,11,14,17).

Nella parte finale del file (linee 21-23) trovano posto eventuali indicazioni riferite a criteri di sicurezza da applicare; nel nostro caso dato che tutta la logica applicativa è gestita dal codice Java realizzato per Virtuose 3.0 è stato deciso di istruire Jetspeed affinché renda visibile a tutti la pagina corrispondente al file psml.

# 5. Conclusioni

Con il lavoro presentato in questa tesi si è giunti alla realizzazione di un prototipo in grado di mostrare come la strada scelta per la realizzazione di Virtuose 3.0 sia percorribile con risultati più che soddisfacenti. La definizione dei nuovi requisiti ha portato ad ampliare le funzionalità del sistema, superando quelle di un gestore di comunità virtuali ed arrivando a fornire anche quelle tipiche di un CMS.

Tutti i requisiti individuati nella fase di progettazione iniziale sono stati soddisfatti e le funzionalità emerse nel corso della progettazione sono state implementate; infatti siamo giunti ad avere un'applicazione con le seguenti caratteristiche:

- in cui i contenuti, memorizzati in messaggi interni alle conferenze, possono essere strutturati sulla base di specifiche (tipi di messaggio) non per forza prefissate ed immutabili ma che sono completamente personalizzabili. Questo fa si che l'amministratore del sistema possa modificare a suo piacimento i tipi già pronti all'uso, possa eliminare quelli che non ritiene realmente utili ed infine possa aggiungerne di nuovi man mano che ne sente la necessità.

  Queste possibilità di intervento rendono Virtuose virtualmente in grado di gestire qualsiasi tipologia di contenuto senza dover riprogettare alcuna sua parte.

- in grado di essere facilmente personalizzata ed estesa, grazie alla sua architettura incentrata sull'uso moduli aventi caratteristiche ben definite per quanto riguarda la propria struttura interna e le funzionalità di interscambio delle informazioni, oltre ad essere facilmente integrabili all'interno del sistema

- in grado di utilizzare molteplici database (rimane comunque il vincolo di averne un solo tipo, deciso in fase di configurazione ed installazione, integrato nel sistema) senza aver la necessità di riprogettare l'architettura di Virtuose o di andare a

modificare tutti i moduli che per il loro corretto funzionamento richiedono i servizi della base di dati; l'unico aspetto del sistema che verrà ad essere modificato è il modulo, con gli eventuali plugin ad esso associati, deputato ad occuparsi delle interazioni dirette con il database. E' da far notare come tutti i moduli per la gestione della base di dati potranno accedere ad essa secondo le proprie modalità ma dovranno assicurare una metodologia comune di dialogo con i restanti elementi di Virtuose, rendendo di fatto del tutto trasparente all'insieme degli altri componenti del sistema l'intercambiabilità dei database.

- capace di permettere l'utilizzo di molteplici fonti di dati relative agli utenti: questo permette di avere potenzialmente una diversa metodologia ed un diverso sistema di gestione per ogni singolo utente, dando modo così di integrare all'interno di Virtuose un numero maggiore di potenziali soggetti attivi sui contenuti, oltre a dar modo di riutilizzare eventuali meccanismi di gestione dell'utenza già attivati per altre applicazioni.

  Anche in questo caso, come in quello della gestione del database, i vari componenti del sistema non strettamente legati alla gestione materiale dei dati degli utenti sono in grado di operare attraverso lo scambio di messaggi in formato standard, restando quindi all'oscuro di quali particolari operazioni vengono eseguite dal plugin specifico per ogni fonte di informazioni.

- la cui creazione e gestione dell'interfaccia grafica e dei contenuti presentati agli utenti è delegata a componenti, basati su XML e tecnologie ad esso legate, che permettono di avere una suddivisione reale tra dati e loro rappresentazione, oltre a fornire possibilità di personalizzazione molto elevate.

  Questo risultato è stato possibile sfruttando dei template di Jetspeed e delle trasformazioni XSLT: attraverso i primi è stata codificato l'aspetto complessivo delle pagine html utilizzate per la creazione dell'interfaccia grafica mentre attraverso le seconde è stata realizzata la vera e propria opera di impaginazione e presentazione

dei contenuti, portandoli dalla loro forma grezza ad una maggiormente indicata per essere fornita agli utenti.

- integrabile con applicazioni già esistenti, risultato ottenuto attraverso la realizzazione di un protocollo di comunicazione, basato su XML ed avente delle specifiche rigorose ma al contempo non limitanti, avente il ruolo di strumento per lo scambio di informazioni tra moduli o tra moduli e altre applicazioni. Oltre a ciò il protocollo è studiato in modo tale da rendere completamente separata la logica applicativa che porta ad una data richiesta/risposta da quello che è il dato vero e proprio scambiato tra le entità ai due estremi del canale di comunicazione; in questo modo per integrare Virtuose all'interno di un'applicazione preesistente è sufficiente realizzare un modulo che si occupi di "tradurre" le richieste fatte dall'applicazione stessa in modo da renderle conformi allo standard interno di Virtuose e che, ovviamente, effettui le eventuali trasformazioni richieste per poter far fare alla risposta ottenuta il cammino inverso.
- basata su tecnologie standard e largamente utilizzate nell'ambito delle applicazioni web: Java per il codice sorgente dell'applicazione, XML per la gestione dei dati e per lo scambio di informazioni tra i vari componenti, XSLT per la trasformazione delle informazioni, organizzate in documenti XML, in modo da creare l'interfaccia utente ed infine l'uso di Portlet per la creazione delle varie porzioni delle pagine html da presentare all'utente
- in grado di essere portato su altre piattaforme hardware e software in modo rapido e senza particolari interventi da parte dell'amministratore: questo risultato è stato raggiunto principalmente grazie alle peculiarità di Java in tale ambito.
- facilmente installabile e configurabile dato che: tutta la logica applicativa e le librerie richieste sono racchiuse in un unico file *war;* un solo file di configurazione facilmente personalizzabile permette di definire il comportamento e le regole su cui basare il funzionamento di tutti i componenti del sistema, la creazione dell'interfaccia utente è

quasi completamente affidata a file *xsl* sui quali non è difficile intervenire.

Per questa prima versione di Virtuose 3.0 è necessario aver installato sulla piattaforma utilizzata per l'esecuzione dell'applicazione il software eXist per la gestione del database e Jetspeed2 per la gestione delle portlet.

- completamente realizzata attraverso l'uso di prodotti rilasciati sotto licenze libere

### *5.1. Sviluppi futuri*

Con questa tesi sono stati realizzati tutti quei componenti di Virtuose 3.0 necessari per la verifica delle idee alla base della sua progettazione e per la messa in opera di un prototipo attraverso il quale dare una dimostrazione pratica delle sue capacità e potenzialità. Esistono però margini di miglioramento e possibilità di sviluppo che possono essere così sintetizzati:

- realizzare un'interfaccia per la creazione,modifica ed eliminazione dei tipi di messaggio in modo da permettere all'amministratore del sistema di gestire questo aspetto fondamentale del sistema senza dover agire direttamente sul database; con ciò si vuole fornire uno strumento più comodo e meno soggetto ad eseguire eventuali operazioni errate, data la presenza della ovvia logica applicativa di controllo

- realizzare l'interfaccia grafica per far inserire,modificare e cancellare in una conferenza un messaggio, permettendo quindi di avere una reale interazione degli utenti sui contenuti presenti nel sistema.

  Particolare attenzione andrà posta sull'inserimento e la modifica dal momento che per tali azioni dovrà essere presentata una maschera ricavata dall'XML-Schema associato al tipo di messaggio su cui operare  e in seguito quanto inserito dall'utente dovrà essere validato, per consentire o meno la prosecuzione dell'azione, sempre secondo tali indicazioni.

Ovviamente dovranno essere tenute in considerazione anche le informazioni circa i permessi contenuti nella vista in uso.

- implementare quelle funzionalità che non sono state ancora rese operative, principalmente si tratta di quelle non strettamente necessarie alla realizzazione della demo e relative a ricerca, inserimento, aggiornamento ed eliminazione dei dati di alcune entità: la parte dei moduli relativa a tali azioni è già presente e nella maggior parte dei casi si tratta solo di realizzare i frammenti XML relativi al codice XUpdate da utilizzare per l'invio dei comandi al database
- inserire nuovi moduli per aumentare le funzionalità del sistema, ad esempio si potrebbe realizzare un insieme di funzionalità utili per la gestione di una mailbox o di un calendario per ciascun utente
- rivedere il file virtuose_jetspeed.psml in modo da rendere la struttura della pagina html da presentare all'utente più piacevole dal punto di vista estetico, in questo caso non ci sono nuove funzionalità da implementare
- creare nuovi template grafici per Jetspeed in modo da distaccarsi da quelli forniti da esso e al contempo raggiungere un livello di presentabilità più consono ad un'applicazione utilizzabile in ambienti non di test
- realizzare nuovi documenti XSLT per migliorare l'aspetto visivo relativo alla presentazione dei dati relativi alle conferenze e ai messaggi, in modo da migliorare l'aspetto delle pagine html, per dare più alternative di visualizzazione per gli stessi dati e per fornire eventualmente spunti su cui basare la costruzione di modelli grafici da applicare a tipologie di messaggi aggiunte al sistema in un secondo tempo

Questo insieme di operazioni volte a portar a compimento Virtuose 3.0 non è ovviamente l'unico attuabile: il prototipo al quale si è giunti con la presente implementazione può infatti essere utilizzato anche come punto di partenza sul quale basare lo sviluppo di nuove applicazioni non espressamente orientate alla gestione di comunità virtuali o alla

presentazione di contenuti, ma che possono trarre vantaggi dalle idee e dalla metodologia scelte per la realizzazione di questa nuova versione di Virtuose.

In quest'ottica le potenzialità di quanto sviluppato nella tesi sono tutte da scoprire.

# Bibliografia


[AJ2] *Apache Jetspeed-2*, http://portals.apache.org/jetspeed-2/

[BDB] *Berkeley DB XML*, http://www.oracle.com/database/berkeley-db.html

[BDS05] Benini M., De Cindio F., Sonnante L. Virtuose, *a VIRTual CommUnity Open Source Engine for Integrating Civic Networks and Digital Cities*, in Proc. Third International Digital Cities Workshop "Digital Cities III:Information Technologies for Social Capital:Cross-Cultural Perspectives", Van den Besselaar P., Koizumi S. (eds.), Lecture Notes in Computer Science, Springer-Verlag, Volume 3081, Pagina 217-232, 2005

[BXML] *Berkeley DB XML*, http://www.sleepycat.com/products/bdbxml.html

[CHI04] Michele Chinosi, *La seconda release di Virtuose basata su database XML*, Tesi, Università degli Studi di Milano, 2004

[SC02] Christopher Schmitt , *CSS Cookbook* , O'Reilly , 2004

[RHE04] Howard Rheingold, *Comunità virtuali*, Sperling & Kupfer Editori, Milano, 1994

[SHE02] Devan Shepherd, *XML*, Apogeo Editore, Pagina 227-248, 2002

[EXI] *eXist: Open Source Native XML Database*, http://exist-db.org/index.html

[FED] Simone Fedele, *Analisi, progetto e realizzazione di database XML: un caso reale*, Tesi di laurea.

[ECK06] Bruce Eckel, *Thinking in Java*, Prentice Hall, 2006

[JDOM] *Java DOM API*, http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/package-summary.html

[JPS] *Introduction to JSR 168 - The Portlet Specification*, http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/

[BAK01] Aneesha Bakharia , *JavaServer Pages: Fast&easy Web Development,* Thomson Course Technology*,* 2001

[OZO] *Ozone/XML*, http://www.ozone-db.org/noframes/ozonies/ozonexml.html

[PSML] *Portal Structure Markup Language*, http://portals.apache.org/jetspeed-2/guides/guide-psml.html

[PSQL] *PostgreSQL*, http://www.postgresql.org/

[SHE02] Devan Shepherd, *XML*, Apogeo Editore, Pagina 249-271, 2002

[VIRT] *Virtuose: Linee Guida*, http://www.virtuose.it/whitepaper.pdf

[VTL] *Velocity Template Language*, http://jakarta.apache.org/velocity/docs/user-guide.html

[XDB] *XML:DB API*, http://xmldb-org.sourceforge.net/xapi/index.html

[XIN] *Apache Xindice*, http://xml.apache.org/xindice/

[YBP04] François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, *Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 4th February 2004*, http://www.w3.org/TR/2004/REC-xml-20040204/

[SIM02] John E.Simpson, *Xpath and Xpointer,* O'Reilly, 2002

[BRU04] Michael Brundage , *XQuery: the XML Query Language* , Addison-Wesley Professional , 2004

[VDV02] Eric van der Vlist , *XML Schema* , O'Reilly , 2002

[VOT02] Michiel Van Otegem*, XSLT* , Apogeo Editore, 2002

[XUPD] *Xupdate*, http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html