



**UNIVERSITA' DEGLI STUDI DI SIENA**  
**FACOLTA' DI INGEGNERIA**

Corso di Laurea In Ingegneria Informatica

*Progettazione di un servizio per  
l'amministrazione e la pubblicazione (web e RSS)  
delle notizie e degli avvenimenti del Comune di  
Grosseto con il framework JSF*

Relatore:

Prof. Marco Maggini

Correlatore:

Dott. Ludwig Bargagli

Tesi di Laurea di:

Carlo Politi

Anno Accademico 2004 – 2005



*Dedicato ai miei cari nonni,  
Ampelio, Gemerina, Luca e  
Giacinta che mi hanno amato e  
sostenuto in tutti questi anni.*

Si ringrazia di cuore: lo staff del S.E.D. del Comune di Grosseto per il supporto datomi durante questi mesi di realizzazione del progetto, Alessandra Pasetto e Manuela Calabria per la loro sopportazione e pazienza nel leggermi la tesi, i miei genitori per avermi sostenuto in questi anni di università ed infine l'Archivio Fotografico della Soprintendenza di Siena nelle persone del Dott. Fabio Torchio e della Dott.ssa Anna Maria Emanuele per la loro amicizia ed affetto in tutti questi anni. Grazie a tutti voi!

# Indice

Dedica e ringraziamenti.....	3
Indice.....	4
Indice delle tabelle.....	5
Indice delle figure.....	6
Capitolo 1	
1.1 Blog su Internet e loro limiti.....	7
1.2 Il Semantic Web e gli standard HTML, XML e RDF.....	9
1.3 Il formato RSS.....	14
Capitolo 2	
2.1 Il formato RSS 2.0 e suo impiego per la diffusione delle notizie del Comune di Grosseto.....	17
2.2 Applicazione client per le notizie uffici del Comune di Grosseto.....	23
2.3 Applicazione client per le notizie riguardanti gli avvenimenti / manifestazioni della provincia di Grosseto.....	37
2.4 Servlets per la diffusione delle notizie del Comune di Grosseto e di quelle riguardanti gli avvenimenti / manifestazioni nella provincia.....	40
2.5 Problematiche riscontrate durante la progettazione e la programmazione dell'applicazione.....	47
2.6 Gli RSS Readers.....	64
Capitolo 3	
3.1 RSS e l'interesse che sta suscitando.....	66
3.2 Podcasting: quando RSS incontra i files audio (e non solo).....	67
3.3 Conclusioni finali.....	69
Capitolo 4	
4.1 Script di generazione delle tabelle per il database Oracle.....	70
4.2 Script di generazione delle tabelle per il database MySQL.....	75
Appendice	
A. Riferimenti.....	80
B. Note finali.....	81

## Indice delle tabelle

I sette formati RSS (Tabella 1).....	15
RSS_GOL (Tabella 2).....	26
RSS_TIPO_NOTIZIE (Tabella 3).....	27
RSS_GOL_CHANNEL (Tabella 4).....	29
RSS_DIRITTI_UTENTI (Tabella 5) .....	30
AREA (Tabella 6).....	30
RSS_GOL_MAIN (Tabella 7).....	37
RSS_AVVENIMENTI (Tabella 8).....	37
MANIFESTAZIONI (Tabella 9).....	38
ANCMOGGI (Tabella 10).....	39
Riepilogo tags formato RSS 2.0 (Tabella 11).....	43
RSS_SEQ_MySQL / RSS_GOL_MAIN_SEQ_MySQL (Tabella 12).....	62

## Indice delle figure

Architettura a tre livelli tipica delle applicazioni Web (Figura 1).....	18
Rappresentazione schematica degli elementi dell'architettura MVC e relativi ruoli (Figura 2).....	21
Architettura Model-View-Controller 2 (Figura 3).....	22
Flusso di controllo dell'architettura Model-View-Controller 2 (Figura 4).....	23
Modello autenticazione Single Sign-On (Figura 5).....	32
Schema semplificato della rete di calcolatori coinvolti nel funzionamento dell'applicazione Web (Figura 6).....	81

# Capitolo 1

## 1.1 Blog su Internet e loro limiti

Il **blog** (parola derivante da **Web log** ossia **diario in Rete**) è un modo per pubblicare o distribuire contenuti nel World Wide Web (o WWW). La grande Rete, cioè Internet, permette a chiunque di costruirsi un proprio sito Web, in poco tempo e a costi ridotti anche se non è difficile trovare siti che offrono spazio gratuito per la costruzione di pagine personali per uso non professionale o commerciale. Esistono molti siti che permettono la realizzazione di blog: uno di questi è, ad esempio, blogger.com. Grazie ai blog, chiunque può mettere on line i contenuti che maggiormente rispecchiano i propri interessi. Nell'era di Internet tutti coloro che hanno qualcosa da comunicare, possono tranquillamente farlo improvvisandosi giornalisti.

L'attacco terroristico agli USA dell' 11 settembre 2001 ha cambiato notevolmente molti aspetti della socialità: tra i cambiamenti rientra anche il giornalismo. Quel tragico evento ha rappresentato uno spartiacque nel mondo del giornalismo on line e ha dato inizio ad un nuovo fenomeno nell'ambito dell'informazione, il **personal journalism**, condotto il più delle volte da non giornalisti. Tale fenomeno si è occupato, sin dalle ore successive alla caduta delle torri gemelle, di raccogliere le testimonianze dei fatti e materiali che riguardano il Ground Zero. In queste testimonianze si trova di tutto, da articoli a cimiteri virtuali, da segnalazioni di "missed" a fotografie, ma sono tutti materiali messi on line da cittadini che sentivano la necessità di condividere col mondo i sentimenti suscitati da quel tragico evento.

Come si vede, il blog è una specie di diario personale pubblico che viene reso visibile a tutti nel Web e, in quanto personale, vengono raccontate impressioni soggettive di chi scrive, eventualmente accompagnando il tutto con immagini o filmati. Grazie poi alla diffusione di macchine fotografiche digitali, sempre più si fanno strada i **photoblog** (ad esempio il sito <http://www.photoblogs.org>); si tratta di blog il cui soggetto sono le fotografie. Un tipico blog usa il testo come sua forma primaria di comunicazione, in un photoblog si dà maggiore enfasi alle fotografie. E' anche vero, però, che alcuni blog contengono immagini, così come certi photoblog contengono pure del testo...ma allora come distinguere un blog da un photoblog?

Quando l'enfasi è la fotografia e le immagini non sono unicamente usate per illustrare il testo, allora possiamo parlare di photoblog.

I blog, come anche i photoblog, hanno un unico autore; alcuni ne hanno anche più di uno e le varie "entries" (o inserimenti) sono di solito pubblicate dall'autore secondo un certo criterio, cioè sono contraddistinte da una data ed un'ora di inserimento e vengono visualizzate in ordine cronologico mettendo in cima alla lista (o in prima pagina) le notizie (o le foto) più recenti così da mettere bene in evidenza l'ultima modifica fatta. Gli inserimenti più vecchi vengono man mano tolti e resi accessibili attraverso dei links che permettono al visitatore del blog di scorrerli cronologicamente: questo significa che esiste un archivio storico organizzato secondo certi criteri quali il mese o la categoria. Molti blog permettono al visitatore di scrivere dei commenti relativi a ciò che è stato pubblicato e questo permette una certa interazione tra l'autore (o gli autori) del blog/photoblog e chi lo visita. E' importante osservare che tutti i blog hanno una natura dinamica che fa pensare che dietro ci sia una qualche forma di **Content Management System** (o **CMS**) piuttosto che una semplice sequenza di codice HTML come nei comuni e statici siti Web. Il CMS, anche se questo termine, ultimamente, è stato associato con i programmi per la gestione dei contenuti, è un sistema usato per organizzare e facilitare la creazione di contenuti digitali nonché di filtrare le notizie più obsolete. Va sottolineato che il blog è adatto per contenuti rapidi oppure per testi che hanno una successione cronologica come i diari, permettono la pubblicazione di avvenimenti o documenti personali; inoltre il blog è molto più facile da gestire rispetto ad un sito personale. Il blog, però, non è adatto per la gestione di contenuti più complessi: infatti il suo maggior limite è sicuramente la mancanza di una organizzazione del contenuto che viene avvertita da chi legge, non certo da chi scrive; infatti, inserire i propri testi nel blog è facile e comodo, invece non lo è recuperarli. Per questo motivo occorre seguire il blog tutti i giorni per riuscire a ritrovare i vari blog. L'unica forma di ordinamento è quella cronologica, cioè vengono mostrate per prime le informazioni più recenti. Non c'è modo di estrapolare direttamente il contenuto, non c'è alcuna struttura navigabile che servirebbe a valorizzare le informazioni, rendendole facilmente accessibili e reperibili anche nel tempo: il visitatore deve necessariamente leggere tutto. Comunque il blog è volutamente così: nasce come un diario, e come tale i suoi contenuti si succedono nel tempo.

## 1.2 Il Semantic Web e gli standard HTML, XML e RDF

Il Semantic Web (o Web semantico) è il futuro del WWW ed è un'iniziativa voluta fortemente da Tim Berners-Lee, inventore del Web e fondatore del W3C, l'organismo che si occupa di scrivere le specifiche e diffondere le tecnologie relative al Web (per riferimento: <http://www.w3.org>). Il W3C fornisce questa definizione di Semantic Web:

*“a common framework that allows **data** to be shared and reused across application, enterprise, and community boundaries”*

Il Semantic Web, quindi, ha a che fare con dati e il suo obiettivo principale è quello di condividere e riutilizzare dati attraverso applicazioni, imprese e comunità. Il problema di accedere alle informazioni e ai contenuti è stato risolto in gran parte dall'invenzione di reti di computer su larga scala, quale il World Wide Web. Il problema di processare ed interpretare l'informazione che è stata recuperata, è ancora un importante argomento di ricerca chiamato **Intelligent Information Integration** (Wiederhold, 1996, Fensel, 1999). I problemi che potrebbero derivare dalla eterogeneità dei dati, sono ben noti alla comunità dei sistemi di database distribuiti e possono essere divisi in tre categorie:

1. Sintassi (esempio: l'eterogeneità del formato dei dati);
2. Struttura (esempio: gli omonimi, sinonimi o differenti attributi nelle tabelle del database);
3. Semantica (esempio: il significato dei termini in un speciale contesto o applicazione).

A livello sintattico la standardizzazione è un aspetto molto importante, molti standard si sono evoluti e possono essere usati per integrare differenti fonti di informazione. In aggiunta alle classiche interfacce per database come l'ODBC, nel corso degli ultimi sono stati sviluppati e hanno alcuni importanti standard orientati al Web quali **HTML**, **XML** e **RDF**.

Iniziamo a parlare del formato HyperText Markup Language (o HTML): la creazione di pagine Web è stata la tecnica più frequente e largamente usata per la condivisione di informazioni. Questo genere di pagine contengono informazioni che hanno testo

sia libero sia organizzato in struttura, immagini e certe volte anche sequenze audio e video. L'HTML è ciò che viene utilizzato per creare questo genere di pagine. Il linguaggio fornisce delle strutture primitive chiamati *tags* che possono essere usate per annotare testo o per incorporare files allo scopo di determinare l'ordine con cui questi elementi dovrebbero essere visualizzati. I tags hanno una sintassi uniforme che viene riconosciuta dai browsers quando questi processano la pagina e generano il layout:

`<nome-tag>informazione(testo)</nome-tag>`

E' importante sottolineare che l'HTML non fa riferimento all'informazione ma si occupa solamente del modo con cui questa deve essere strutturata e rappresentata nella pagina: questo è anche il suo più grande svantaggio perché il processo di comprensione e valutazione del contenuto offerto è lasciato interamente all'utente. Per risolvere questo problema, per offrire l'accesso all'informazione pur mantenendo la "libertà" di dire qualsiasi cosa circa un particolare argomento, è stato introdotto un nuovo standard: l'eXtensible Markup Language (o XML). L'HTML non permette di definire strutture dati, ha uno schema di definizione fisso e per superare questa limitazione, l'XML è stato proposto come linguaggio estensibile che consente all'utente di definire i suoi propri tags in modo da evidenziare il tipo di contenuto riferito dal tag. Il principale vantaggio dell'XML risiede nella possibilità di scambiare dati in modo strutturato e con l'introduzione degli schemi XML si è cercato di dare una definizione di linguaggio per le strutture dati per enfatizzare questa idea. Cerchiamo ora di presentare l'idea che sta dietro all'XML e di descrivere le definizioni di schemi e le loro potenzialità per lo scambio di dati. Possiamo parlare di documento XML quando questo oggetto rispetta le linee guida del consorzio del W3C per la scrittura di un documento ben formato: queste specifiche forniscono la grammatica formale che va usata e in aggiunta a queste, l'utente può imporre ulteriori vincoli grammaticali alla struttura del documento usando un **document type definition** (o **DTD**). Un documento XML è dunque valido se ha associato un tipo di definizione e ne segue i vincoli grammaticali. Il DTD invece specifica gli elementi che possono essere usati all'interno del documento XML: questi elementi sono sempre delimitati da un tag iniziale e da un tag finale. I vincoli presenti nel DTD si riferiscono alla struttura logica del

documento e questo include anche le ramificazioni dei tags all'interno del corpo dell'informazione che sono consentite e/o richieste. Ulteriori restrizioni possono essere espresse nel DTD circa il tipo di attributi e i valori di default che possono essere usati quando non è stato fornito alcun valore all'attributo. Uno schema XML è di per sé un documento XML che definisce la struttura valida di un documento XML secondo lo spirito del DTD e gli elementi utilizzati nello schema di definizione sono di tipo "elemento" e hanno attributi che definiscono le restrizioni già menzionate. L'informazione all'interno di un elemento è semplicemente una lista di ulteriori definizioni di elementi che devono essere annidati all'interno di quello specifico elemento:

```
<elemento nome="valore" tipo="valore" ...>  
<elemento nome="valore" occorrenzaMinima="valore" .../>  
...  
</elemento>
```

In aggiunta, lo schema XML ha altre "features" (o caratteristiche) molto utili per la definizione delle strutture dati:

- Supporto per i tipi base di dati;
- Vincoli sugli attributi (esempio: vincoli sulle occorrenze);
- Strutture sofisticate (esempio: definizioni derivate estendendo o restringendo altre definizioni);
- Un meccanismo di **namespace** (cioè di un contesto che definisce un insieme di tags da utilizzare in documento XML) che consente la combinazione di differenti schemi.

Non discuteremo in dettaglio queste caratteristiche ma bisogna citare la possibilità che le features aggiuntive rendano possibile la codifica di strutture dati piuttosto complesse. Questo ci permette di tracciare il modello dati dell'applicazione, la cui informazione la vorremmo condividere con altri attraverso uno schema XML. Una volta tracciato il modello, possiamo codificare la nostra informazione in termini di documento XML e renderlo (combinato con uno schema XML) disponibile su Internet. Lo scambio di informazione è mediata attraverso differenti formati nel seguente modo:

## Application Data Model ↔ XML schema → XML document

Questo metodo ha una grande potenzialità per l'effettivo scambio di dati ma l'utente deve interagire con il modello dati per poter far uso dell'informazione. Come è già stato precedentemente detto, uno schema XML definisce la struttura dati e non fornisce nessuna informazione circa il contenuto o le potenzialità di utilizzo dell'informazione e questa mancanza è stata colmata introducendo un nuovo linguaggio: il Resource Description Format (o RDF). Questo standard è stato proposto come un data-model per la rappresentazione dei meta-dati (cioè dati che descrivono altri dati) delle pagine Web e del loro contenuto usando una sintassi XML. Il principio di base dell'RDF è semplice perché ogni tipo di informazione circa una specifica risorsa, che possa essere una pagina Web o un elemento XML, è espresso in termini di triple:

(risorsa, proprietà, valore)

C'è dunque una relazione che collega una risorsa ad un certo valore, che può essere un semplice dato o il nome di un'altra risorsa, di tale proprietà. In aggiunta, il valore può essere rimpiazzato da una variabile che rappresenta una risorsa successivamente descritta mediante il collegamento a delle triple che descrivono le proprietà della risorsa rappresentata da quella variabile:

(risorsa, proprietà, X)

(X, proprietà\_1, valore\_1)

...

(X, proprietà\_n, valore\_n)

Un'altra feature dell'RDF è il meccanismo che rende possibile l'utilizzo di una tripla come valore di una certa risorsa, ad esempio:

(risorsa\_1, proprietà\_1, (risorsa\_2, proprietà\_2, valore))

Un ulteriore problema che deriva dalla natura del Web è la necessità di eliminare i conflitti che potrebbero verificarsi quando si fa riferimento a siti Web che potrebbero usare differenti modelli di RDF per descrivere i meta-dati ma per ovviare a questo

problema, l’RDF utilizza i namespace che sono forniti dall’XML. Questi sono definiti una volta per tutte attraverso un riferimento ad un **Uniform Resource Identifier** (o **URI**) che fornisce i nomi che saranno uniti ad un “source-ID” che è poi usato per annotare ciascun nome in una descrizione RDF che definisce l’origine di tale particolare nome nel seguente modo:

source\_id:nome

E’ stata inoltre realizzata una sintassi standard per definire gli statements RDF che rendono possibile l’identificazione degli statements come meta-dati.

Dopo questa carrellata circa i tre principali standard che si sono sviluppati ultimamente, torniamo a parlare del Semantic Web. Il Web, come molti sanno, consente di condividere e riutilizzare documenti in formato HTML e lo possiamo paragonare ad una grande biblioteca in cui gli utenti sono le persone che cercano e leggono testi. Dobbiamo quindi pensare al Semantic Web come ad un grande database mondiale in cui questi “utenti” sono programmi o agenti software che aiutano le persone fisiche a gestire e selezionare i dati di loro interesse. Nel febbraio 2003, il W3C ha rilasciato le specifiche delle tecnologie alla base del Semantic Web: Resource Description Framework (RDF), RDF Schema e Web Ontology Language (OWL) utilizzati per descrivere le risorse e i dati che saranno disponibili in maniera comprensibile alle macchine (o a speciali applicazioni software). Di recente è iniziata la seconda fase di diffusione del Semantic Web che ha come obiettivo la creazione di database e applicativi scalabili in grado di gestire dati secondo lo standard RDF e questo sta portando alla creazione di applicazioni software che affrontano il tema della gestione dei dati in una modalità totalmente nuova e funzionale. Facendo un paragone, come l’XML da un punto di vista di sintassi ha permesso di integrare sistemi e applicazioni differenti, così il Semantic Web con l’RDF consentirà l’integrazione “semantica” dei dati. I settori che trarranno maggiore beneficio dall’utilizzo di queste tecnologie saranno quello dell’Enterprise Information Integration, del Knowledge Management e dell’Information Retrieval tramite l’utilizzo di meta-dati e di ontologie. Una prima applicazione di concreta utilità va sotto il nome di **RDF Site Summary (RSS)** o anche **syndication**. Molti weblog così come i siti di news esportano i loro contenuti in formato XML facendo uso di un vocabolario con una semantica ben precisa, quella dell’RSS appunto. Tutto ciò

richiede la necessità di costruire applicativi che siano in grado di leggere, o meglio, di aggregare e distribuire notizie e contenuti tra siti differenti, portali od organizzazioni.

### 1.3 Il formato RSS

Cerchiamo ora di capire cosa significa la sigla RSS e cosa si nasconde dietro. Per prima cosa, diamo una definizione di tale sigla:

*RSS è un formato per la distribuzione di contenuti sul Web*

E' una definizione generica ma che lascia spazio a sviluppi futuri del formato. Come linguaggio, l'RSS è una applicazione di XML e questo ci porta a dire che deve necessariamente essere conforme alla specifica XML 1.0 (<http://www.w3.org/TR/REC-xml/>) oltre ad essere ben formato. Quale è l'utilizzo concreto dell'RSS? Consideriamo di avere una pagina Web che tratta un particolare argomento e vogliamo far sapere, a chi ne è interessato, di ciò che trattiamo: se creiamo un elenco degli argomenti trattati e li mettiamo come "item" nel nostro file RSS, chi si collegherà a tale file riuscirà a sapere costantemente cosa abbiamo aggiornato e questo grazie all'utilizzo di uno speciale lettore di RSS, una specie di "programma di posta elettronica" che va sotto il nome di **news aggregator** (o **aggregatore**). L'RSS risolve uno dei problemi delle persone che usano regolarmente il Web, cioè quello di essere costantemente aggiornati su i cambiamenti all'interno di una certa pagina di interesse attraverso il recupero del sommario di ciò che è stato aggiornato. In questo modo dunque si guadagna tempo perché non è più necessario visitare interamente tutto il sito per accorgersi di ciò che è stato variato. Un altro problema che L'RSS risolve è la garanzia della privacy perché l'RSS non è una mailing-list o una newsletter e quindi non si deve fornire alcun dato, tanto meno l'indirizzo e-mail. Il numero di siti che offrono feed RSS sta aumentando rapidamente come ad esempio il sito di Yahoo News (<http://news.yahoo.com/rss/>) o quello di Amazon.com (<http://www.amazon.com/exec/obidos/subst/xs/syndicate.html>), tanto per citare i più famosi a livello internazionale. Cosa è un feed reader o un news aggregator? E' un particolare software che periodicamente recupera i feed RSS da diversi siti e

consente all'utente di essere informato sugli aggiornamenti di un certo argomento. Quando nasce però l'idea di RSS? Come vedremo, questo nome fa da copertura per un formato che ha dato origine a diverse versioni, di cui almeno due differenti formati anche se nati in parallelo. Nel marzo 1999, la versione originale di questo protocollo, la 0.90, fu progettata da Netscape con lo scopo di costruire portali di titoli di articoli ma il progetto era ritenuto troppo complesso. Venne così proposta la versione 0.91 che era molto più semplice ma successivamente la Netscape perse interesse nell'attuazione del suo progetto di creazione di portali: tuttavia, questa versione fu continuata da un'altra ditta, la UserLand, che si propose di usarla come base per i suoi prodotti di weblog e per la scrittura di altri software web-based. Nel frattempo, un terzo gruppo, con fini non commerciali, ideò un nuovo formato basato su ciò che aveva notato come gli originali principi guida della versione 0.90 (quindi prima che fossero semplificati nella 0.91). Questo nuovo formato, basato sullo standard RDF, è stato chiamato RSS 1.0 ma non è stato sviluppato in collaborazione con UserLand che, non accettando questa versione, continuò a sviluppare un suo filone producendo le versioni 0.92, 0.93, 0.94 e l'attuale 2.0. Con tutti questi formati, ben sette, quale versione va scelta per la distribuzione dei contenuti e quali sono quelli che devono essere usati? Riassumiamo le caratteristiche (**Tabella 1**):

<b>Versione</b>	<b>Sviluppatore</b>	<b>Vantaggi</b>	<b>Status</b>	<b>Raccomandazioni</b>
<b>0.90</b>	Netscape		Obsoleto e rimpiazzato dalla 1.0	Da non usare più
<b>0.91</b>	UserLand	Versione semplificata della 0.90	Ufficialmente obsoleto, rimpiazzato dal 2.0 ma ancora abbastanza popolare	Da usare per scopi semplici. Facile migrazione alla versione 2.0 per una maggiore flessibilità
<b>0.92, 0.93, 0.94</b>	UserLand	Consente una maggiore ricchezza di meta-dati rispetto alla 0.91	Obsoleto e rimpiazzato dalla 2.0	Usare la versione 2.0
<b>1.0</b>	Gruppo di lavoro RSS-Dev	Basato su RDF, estensibile attraverso moduli, non	Core stabile, sviluppo attivo di moduli	Usare per applicazioni basate su RDF o se è necessaria un'avanzata

		gestito da un unico sviluppatore		gestione dei moduli RDF
<b>2.0</b>	UserLand	Estensibile attraverso moduli, facile migrazione dalle versioni 0.9x	Core stabile, sviluppo attivo di moduli	Da usare per scopi multiuso, ricco utilizzo di meta-dati

Tabella 1

## Capitolo 2

### 2.1 Il formato RSS 2.0 e suo impiego per la diffusione delle notizie del Comune di Grosseto

Nel precedente capitolo abbiamo visto come il bisogno di recuperare informazioni dalle pagine Web ha portato alla creazione di nuovi standard e in particolare al formato (o ai vari formati) RSS. In questo capitolo descriveremo come è composto un file del genere e come è stata realizzata un'applicazione Java per la gestione e divulgazione dei feed RSS per le notizie e gli avvenimenti del Comune di Grosseto. Abbiamo già accennato che i files RSS sono conformi allo standard XML 1.0 e contengono una lista (o feed) di titoli, sommari e links recuperabili da uno speciale lettore, detto appunto news aggregator, RSS reader o, più semplicemente, aggregatore. Abbiamo già accennato che esistono ben sette versioni di RSS e questo porta ad un dilemma per decidere quale sarà la versione o il formato universalmente valido, ma dovendo fare una scelta per questo progetto, abbiamo deciso di attenerci alle specifiche della versione 2.0 (<http://blogs.law.harvard.edu/tech/rss>) perché è la versione più recente e più semplice da usare e si presta bene per un utilizzo di carattere generale e perché ha permesso lo sviluppo di nuovi modi di diffondere informazione come vedremo nel prossimo capitolo. Partiamo dalle motivazioni che hanno spinto alla realizzazione di questo servizio per la diffusione delle notizie per il Comune di Grosseto e cioè quello di rimpiazzare la loro vecchia procedura di diffusione di notizie operante in PHP. I vincoli che dovevano essere rispettati erano di mantenere la compatibilità con il rullo scorrevole delle notizie comunali e con i dati dell'archivio collegato con la form di ricerca degli avvenimenti, operante su database Oracle 10g 9.0.4. Partiamo col dire quali sono stati i nostri "ingredienti". L'intero applicativo è stato scritto in linguaggio Java in ambiente Enterprise (J2EE 1.4, J2SE 1.4.2). Per lo sviluppo è stato utilizzando il nuovo framework JavaServer Faces 1.2 (JSF) e il tutto è stato sviluppato con Oracle JDeveloper 10g 10.1.3. Il nostro intento era anche quello di creare una procedura che fosse compatibile sia con prodotti commerciali sia con prodotti Open Source quindi abbiamo reso disponibile un'unica versione per gli application server: Oracle 10g (9.0.4.1 e 10.1.2), Oracle OC4J (10.1.2 e 10.1.3), Apache Jakarta Tomcat 5.5 e per i database: Oracle 10g 9.0.4 e MySQL 4.1. E' stato anche scelto di utilizzare la Java Virtual Machine (JVM)

versione 1.4.2 invece della più recente 1.5: abbiamo infatti riscontrato delle differenze sostanziali tra le due versioni a livello di utilizzo di certi metodi che erano stati in parte aggiornati. L'aver preferito la versione più vecchia a quella più recente, ci ha permesso di evitare delle incompatibilità a livello di librerie sia con l'ambiente di sviluppo da noi utilizzato (Oracle JDeveloper 10g 10.1.3) sia con i vari application server su cui abbiamo provato a far girare la nostra applicazione. Va chiarito che per poter far funzionare Apache Jakarta Tomcat 5.5 è stato necessario installare la versione 1.5 della JVM benché l'intero progetto fosse compilato con la 1.4.2. Prima di iniziare ad illustrare il progetto svolto, iniziamo ad introdurre le applicazioni Web e quale architettura abbiamo scelto. Quando si scrivono software il cui target è il Web, come nel nostro caso, parliamo di un tipo di applicazioni che ha certe caratteristiche distintive quale l'uso di un'architettura a tre livelli (**Figura 1**), in cui tra il database e il client, rappresentato dal browser Web, viene inserito un elemento intermedio, chiamato **middle tier**. Quest'ultimo ha il compito di trasformare richieste espresse secondo il protocollo HTTP in richieste verso il sistema informativo e, viceversa, convertire i risultati delle interrogazioni in pagine visualizzabili dal client; funge dunque da **gateway**, i cui compiti appartengono sommariamente a due tipologie: da un lato la cosiddetta logica di business, cioè l'insieme delle funzionalità necessarie per l'evasione delle richieste dell'utente; dall'altro formattare i risultati computati per la visualizzazione delle funzioni di business, ad esempio sotto forma di pagine ipertestuali in HTML. Quanto più questi due compiti del gateway risultano separati tanto più le funzioni di business e le logiche di presentazione sono riutilizzabili e possono venire sfruttate per generare interfacce personalizzate, per esempio su dispositivi diversi, tra cui PC, PDA, telefoni cellulari, o altro ancora. La programmazione delle funzioni del middle tier è uno degli aspetti più complessi dello sviluppo di un'applicazione Web, poiché è a questo livello che si concentrano gran parte delle funzionalità.

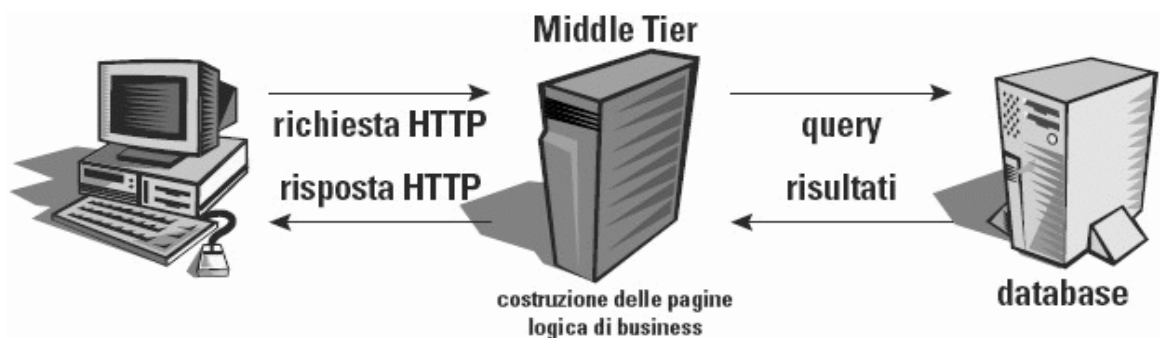


Figura 1

Le righe di codice che seguono rappresentano un esempio di pagina scritta in Java Server Pages (JSP) che hanno il compito di tradurre una richiesta dell'utente in una pagina HTML. In questo esempio sono presenti tutti gli aspetti della programmazione del middle tier e cioè:

- **la logica di accesso ai dati**, cioè la parte di codice che si occupa della connessione alla base di dati e della formulazione delle interrogazioni;
- **la logica di presentazione**, che si occupa di produrre i taggs HTML necessari per la visualizzazione;
- **la logica di controllo**, che determina il comportamento dell'applicazione a fronte dell'interazione dell'utente; nel caso presente, tale logica riguarda la costruzione dell'indirizzo della pagina da mostrare quando l'utente seleziona una voce dall'indice (**Dettaglio.jsp**).

```
<%@page language="java" %>
<%@ page import="java.sql.*" %>
<!--Logica di accesso ai dati -->
<%
// Connessione al database
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn =
DriverManager.getConnection("jdbc:odbc:acme","acmeuser", "acmepwd");

// Preparazione ed esecuzione della query
Statement stmt = conn.createStatement();
ResultSet result = stmt.executeQuery("select codice,nome from titoli order by
codice");
%>
<!-- Logica di presentazione -->
<!-- Produzione del contenuto HTML -->
<html>
  <head>
    <title>Elenco dei titoli</title>
  </head>
  <body>
    <!-- Produzione dell'indice -->
    <table>
      <% while (result.next()) { %>
      <tr>
        <!-- Produzione delle ancore ipertestuali -->
        <td>
          <!-- Logica di controllo -->
          <a
            href="Dettaglio.jsp?ID=<%=result.getString("codice")
            %>">
```

```

                                <%= result.getString("nome") %></a>
                                </td>
                                </tr>
                                <% } %>
                                </table>
                                </body>
                                </html>
                                <%
                                result.close();
                                stmt.close();
                                conn.close();
                                %>

```

Da questo esempio possiamo concludere che la concentrazione di tutte e tre le componenti di logica in un unico modulo rende più difficile il riutilizzo e la manutenzione evolutiva, non è possibile apportare modifiche ad un solo aspetto indipendentemente dagli altri, ad esempio non è consentito cambiare la pagina di output senza intervenire sull'intero template JSP. Considerazioni simili a queste hanno indotto i progettisti di architetture software a studiare tecniche di organizzazione dei componenti del middle tier in grado di migliorare la separazione degli aspetti e garantire una migliore manutenzione e scalabilità. Una tra le soluzioni proposte che ha riscosso maggiore successo prevede l'utilizzo di **Presentation Framework** basati sullo schema **Model-View-Controller (MVC)**. L'idea dei Presentation Framework MVC non nasce con le applicazioni Web, poiché il concetto di *presentation framework* è una particolare incarnazione dell'idea più generale di *software framework*, nato nel mondo della programmazione a oggetti. Un software framework è definibile come un insieme di componenti software (per esempio classi e/o interfacce) che forniscono una soluzione parziale, ma altamente riutilizzabile, ad un problema generale che deve essere completata dal progettista con ulteriori componenti specifici. Un framework fornisce l'impalcatura della soluzione, detta i requisiti di funzionamento dei componenti e gestisce nel modo migliore la loro interazione, evitando quelle forme di interazione diretta tra i componenti che ne impediscono il riuso. Ciò che viene chiesto al programmatore è di scrivere metodi che implementino le funzionalità richieste dal framework, cui è lasciato il compito di invocare ciascun componente al momento giusto. MVC prescrive la tripartizione delle funzioni di un'applicazione interattiva secondo lo schema mostrato in **Figura 2** e detta le responsabilità dei vari partecipanti:

- Il **Model** rappresenta lo stato dell'applicazione, per esempio i principali oggetti di un dominio applicativo;
- La **View** visualizza nel modo opportuno i contenuti del Model, registrandone le variazioni;
- Il **Controller** traduce le azioni effettuate dall'utente sulla vista in aggiornamenti dei contenuti del Model.

In base all'interazione dell'utente e all'esito delle azioni scatenate sul Model, il Controller seleziona la View da utilizzare per formulare la risposta all'utente.

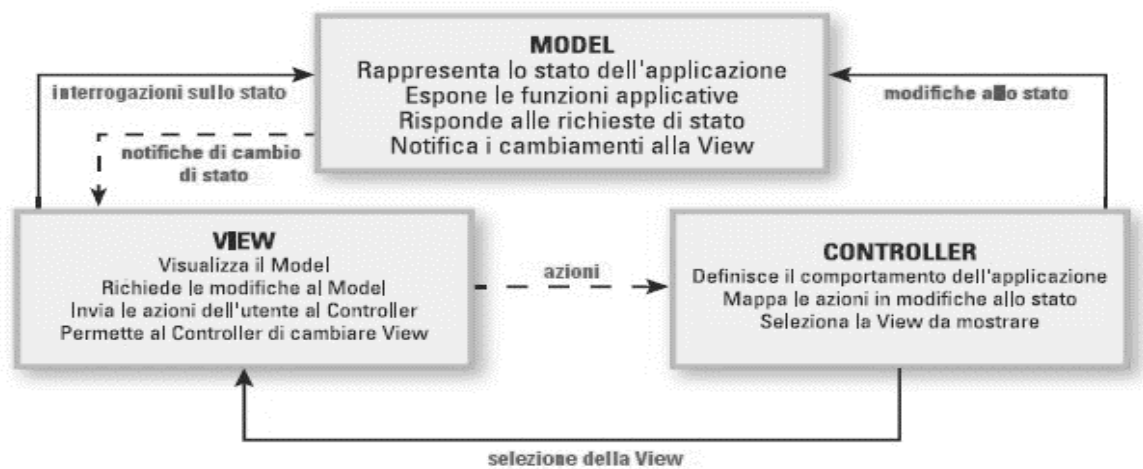


Figura 2

Lo schema MVC garantisce un miglior isolamento tra i vari componenti, poiché:

1. Le azioni dell'utente sono indirizzate unicamente al Controller, unico responsabile della decisione su come formulare la risposta;
2. Il Model non dipende dalla View, per cui può essere reso mediante View differenti;
3. La View non dipende dalle modalità di aggiornamento del Model e non è influenzata da eventuali modifiche della logica con cui quest'ultimo viene aggiornato.

Abbiamo accennato che MVC nasce nel contesto della programmazione a oggetti tradizionale: esso viene adattato alle architetture Web, in particolare alla piattaforma Java 2 Enterprise Edition (J2EE), tenendo presente la mancanza in HTTP di meccanismi per mantenere lo stato dell'interazione e per notificare al client cambiamenti insorti a lato server. Il risultato di tale adattamento va sotto il nome di

architettura MVC 2 (**Figura 3**) ed è anche quella che abbiamo utilizzato nella nostra applicazione Web.

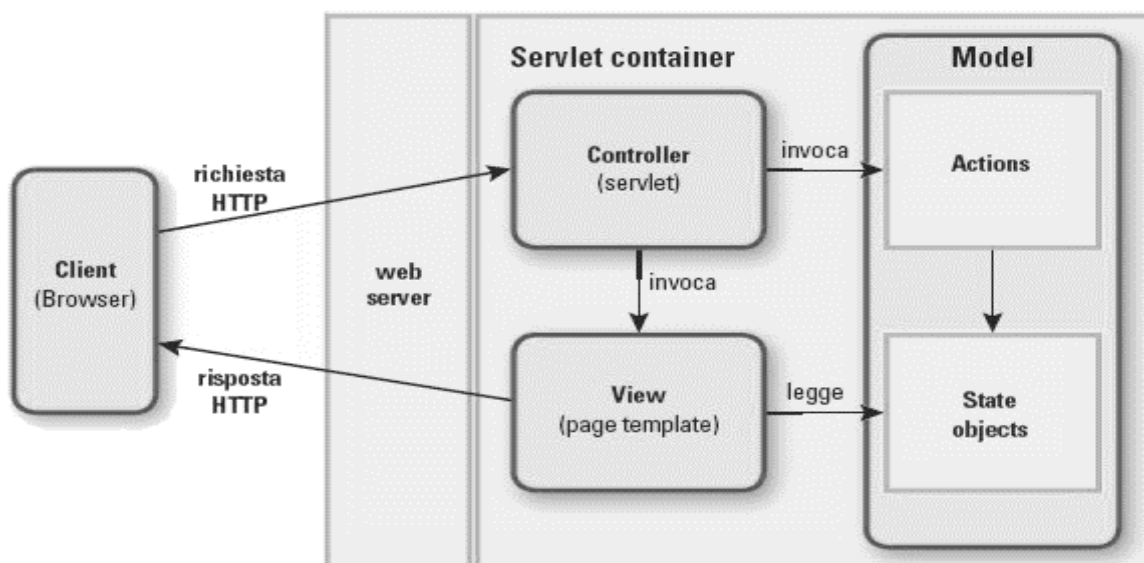


Figura 3

In questo tipo di architettura il browser è il generatore delle richieste: quando l'utente attiva un collegamento ipertestuale nella pagina HTML o usa una form, la richiesta HTTP viene indirizzata verso un'unica servlet che svolge le funzioni di Controller. E' il Controller che decide la sequenza di azioni necessarie per soddisfare la richiesta; le azioni ammissibili sono rappresentate sotto forma di componenti object-oriented, chiamati **action class**. Nello specifico, il Controller associa le richieste HTTP alla corrispondente azione creando o riutilizzando un oggetto della action class e chiamando una funzione predefinita. Ogni action class racchiude una particolare funzione applicativa che interroga e/o modifica lo stato dell'applicazione. Nella situazione più semplice, una action contiene tutta la logica necessaria a servire la richiesta HTTP; in quelle più complesse la classe action svolge unicamente un ruolo di intermediazione nei confronti di altri oggetti, appartenenti al Model, che contengono la logica di business vera e propria. Esempi di azioni eseguite o mediate da una action class possono essere l'esecuzione di un'interrogazione di una base di dati, l'invio di messaggi di posta elettronica o anche l'autenticazione dell'utente: alla fine dell'esecuzione, l'azione comunica l'esito al Controller, che decide il passo successivo. Tipicamente il Controller invoca un template JSP che fa parte della View. Il template JSP accede agli oggetti del Model che contengono lo stato corrente dell'applicazione e costruisce una pagina (ad esempio HTML) che visualizza tale

stato all'utente. La **Figura 4** riassume quanto detto circa la sequenza delle interazioni tra i componenti dell'architettura MVC 2.

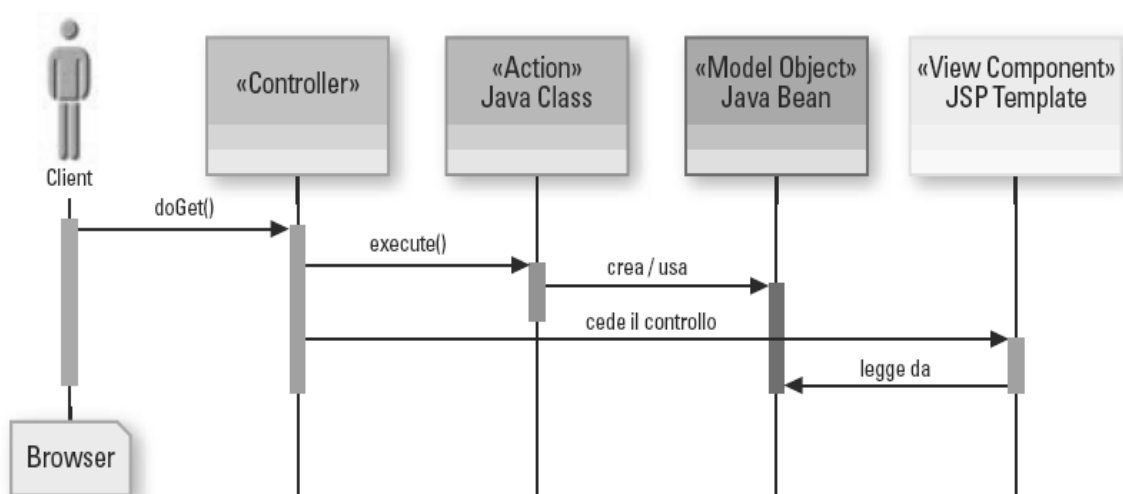


Figura 4

Dopo aver dato un'idea del tipo di architettura Web da noi utilizzata e come si articolano i vari componenti, iniziamo ad analizzare la struttura del nostro progetto, vediamo che è composto di due parti: una riguardante l'amministrazione, tramite la quale l'utente può effettuare inserimenti, modifiche e ricerche di notizie; l'altra ha il compito di recuperare dal database le informazioni offrendole come feed RSS attraverso l'uso di servlet. Per l'esattezza, l'applicazione che abbiamo sviluppato è formata da ben due procedure di gestione delle notizie: una per quelle specifiche dei vari uffici e l'altra per quelle riguardanti gli avvenimenti / manifestazioni presenti nella provincia di Grosseto. Stesso discorso va fatto per le due servlet che sono incaricate di generare il codice XML del feed RSS. Tratteremo queste parti singolarmente e, dove possibile, le accorperemo mostrando le eventuali somiglianze.

## 2.2 Applicazione client per le notizie degli uffici del Comune di Grosseto

L'applicazione client che permette all'utente di fare inserimenti, ricerche e correzioni dei dati memorizzati nel database delle notizie è composta da varie pagine **Java Server Pages (JSP)**. Questo tipo di tecnologia permette di fondere codice HTML statico con contenuto generato dinamicamente da servlet. Le pagine Web che sono costruite mediante programmi **Common Gateway Interface (CGI)** sono principalmente statiche, con parti che cambiano limitatamente a poche locazioni. Per

esempio, consideriamo il caso della pagina iniziale di un qualche negozio on-line (o altro servizio): generalmente tale pagina è uguale per tutti i visitatori, fatta eccezione per quando vi accede un utente riconosciuto (magari mediante un precedente processo di autenticazione). La maggior parte delle varianti CGI però, includendo le servlets, permettono di generare l'intera pagina mediante il proprio programma, anche se la maggior parte della pagina è sempre la stessa. La tecnologia JSP permette di creare le due parti separatamente. Il pezzo di codice qua riportato illustra come realizzare una ipotetica pagina introduttiva di un negozio on-line mediante uso di JSP in cui la parte generata dinamicamente (parte in neretto) è marcata con speciali tags simili a quelli usati dall'HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
  <H1>Welcome to Our Store</H1>
  <SMALL>Welcome,
  <!-- User name is "New User" for first-time visitors -->
  <%= Utils.getUserNameFromCookie(request) %>
  To access your account settings, click
  <A HREF="Account-Settings.html">here.</A></SMALL>
<P>
  Regular HTML for all the rest of the on-line store's Web page.
</BODY>
</HTML>
```

La tecnologia JavaServer Pages offre molte molti vantaggi rispetto ad altre sue ben note alternative:

- Nel confronti di **Active Server Pages (ASP)** di Microsoft i vantaggi sono duplici. Per prima cosa, la parte dinamica di una pagina viene scritta in Java perché più robusto e adatto ad applicazioni complesse, dove si richiede l'uso di componenti riutilizzabili. Inoltre si rileva la portabilità di JSP verso altri sistemi operativi e Web servers poiché non si è chiusi dentro sistemi Windows NT/2000 con Internet Information Services (IIS); con JSP si può usare Java e non essere legati ad uno specifico prodotto server;
- **PHP** è un linguaggio script di tipo HTML-embedded gratuito e Open Source che è in qualche modo simile sia ad ASP sia a JSP. Il vantaggio di JSP in questo caso è che le parti dinamiche sono realizzate in Java, che probabilmente già si conosce, che possiede esaurienti API per il networking,

accesso ai database, oggetti distribuiti mentre con PHP si è costretti ad imparare un linguaggio totalmente nuovo e diverso;

- JSP non fornisce alcuna possibilità che non potrebbe in principio essere compiuta mediante una **servlet**. I documenti JSP sono infatti automaticamente convertiti in servlet in modo trasparente all'utente: è più conveniente scrivere, nonché modificare, una pagina HTML che avere moltissime istruzioni "println" che generano il codice HTML. In più, mediante la separazione della rappresentazione dal contenuto, è possibile far lavorare più persone sullo stesso compito: gli esperti di Web design possono realizzare la parte HTML con i loro specifici tools e lasciare il posto ai programmatori che inseriranno il codice per la parte dinamica;
- **Server-Side Includes (SSI)** è una tecnologia largamente supportata per inserire pezzi di codice, definiti esternamente, all'interno di pagine Web statiche. Anche in questo caso JSP risulta migliore perché offre un set più ricco di tools per la costruzione di tale pezzo esterno di codice e ha più opzioni riguardanti lo stadio della risposta HTTP a cui il pezzo viene effettivamente inserito. In aggiunta, SSI è adatto solo per semplici inclusioni o brevi frammenti di codice, non per reali e complessi programmi che usano form di dati o creano connessioni a database;
- Il **JavaScript**, che è completamente distinto dal linguaggio di programmazione Java, è normalmente utilizzato per generare dinamicamente HTML sul client, costruendo parti di pagine Web non appena il browser carica il documento. Questa è una capacità utile ma gestisce solo situazioni dove l'informazione dinamica è basata sull'environment del client. Fatta eccezione dei cookies, la richiesta dati HTTP non è disponibile alle routine JavaScript lato client; non è possibile realizzare codice JavaScript sul lato client che acceda alle risorse lato server tipo database poiché tale linguaggio non dispone di routine per la programmazione orientata alle reti;
- L'**HTML** non può contenere informazioni dinamiche e perciò le pagine HTML statiche non possono essere basate su input dell'utente o su sorgenti di dati del lato server. JSP è così semplice ed utile che è piuttosto ragionevole arricchire le pagine HTML mediante l'inserimento di dati dinamici.

Ora che è stata data un'idea di cosa permetta di fare la tecnologia JSP, possiamo descrivere la struttura del nostro database e di alcune tabelle: nel seguito, faremo riferimento alle tabelle create con Oracle 10g 9.0.4 ma questo solo per comodità di esposizione. Dove necessario verrà evidenziata la differenza tra Oracle e MySQL e saranno poi dati gli script per la generazione delle tabelle per i due diversi database. Per prima cosa consideriamo la tabella **RSS\_GOL (Tabella 2)** che ha un vincolo referenziale dal campo *TIPO\_NOTIZIA* al campo *ID* della tabella **RSS\_TIPO\_NOTIZIE (Tabella 3)** di cui daremo dopo la descrizione. Solo tre campi in questa tabella devono essere obbligatoriamente popolati: il campo *TITLE* che contiene il titolo della notizia; il campo *DESCRIPTION* che contiene il testo della notizia e ovviamente *ID* che è una chiave primaria. Poniamo l'accento sul campo *PUBDATE* che ci permette di tenere traccia della data e dell'ora dell'inserimento della notizia e sottolineiamo subito la differenza che c'è nell'uso di questi campi utilizzando Oracle e MySQL. Nel primo caso, i campi data/ora sono identificati dal tipo *Date*: questi permettono di memorizzare sia informazioni sulla data sia sull'orario, delegando al programmatore il compito di scegliere quale delle due informazioni memorizzare o utilizzare. Nell'altro caso invece, quello con MySQL, ci troviamo davanti ad una divisione di tipi: i campi per la data sono gestiti da campi *Date* e quelli per l'orario dai campi *Time*. Esiste però in MySQL il modo di tenere traccia della data e dell'ora allo stesso momento facendo uso del campo di tipo *Timestamp*: avremmo potuto anche far uso del tipo *Datetime* benché la principale differenza tra i due tipi è quella che il primo aggiorna il suo contenuto con la data e l'ora in cui viene fatto o un inserimento o un aggiornamento, in via del tutto automatico, mentre il secondo delega il programmatore ad inserire manualmente le due informazioni. Dovendo realizzare un'applicazione che fosse compatibile sia con Oracle 10g sia con MySQL e avendo utilizzato per quest'ultimo il tipo *Timestamp*, è stato necessario utilizzare dei metodi compatibili ad entrambi per accedere alle informazioni sulla data e sull'orario. Abbiamo scelto di utilizzare il metodo *getTimestamp*, appartenente alla classe *java.sql.ResultSet*, per la lettura e *setTimestamp*, appartenente alla classe *java.sql.PreparedStatement*, per la memorizzazione dei campi data/ora.

<i><b>Campo</b></i>	<i><b>Tipo</b></i>	<i><b>NULL</b></i>	<i><b>Chiave</b></i>
TITLE	VARCHAR2(150 BYTE)		
LINK	VARCHAR2(100 BYTE)	SI	
DESCRIPTION	VARCHAR2(4000 BYTE)		

CREATOR	VARCHAR2(8 BYTE)	SI	
CATEGORY	VARCHAR2(40 BYTE)	SI	
COMMENTS	VARCHAR2(250 BYTE)	SI	
GUID	VARCHAR2(100 BYTE)	SI	
PUBDATE	DATE	SI	
SOURCE	VARCHAR2(100 BYTE)	SI	
DIREZIONE	NUMBER(5)	SI	
TIPO_NOTIZIA	NUMBER(3)	SI	MUL
ENCLOSURE_URL	VARCHAR2(100 BYTE)	SI	
ENCLOSURE_LENGTH	NUMBER(7)	SI	
ENCLOSURE_TYPE	VARCHAR2(20 BYTE)	SI	
ID	NUMBER(7)		PRI

Tabella 2

La tabella **RSS\_TIPO\_NOTIZIE** (**Tabella 3**) contiene i tipi di notizie che possiamo inserire, ad esempio “Informazioni sito” oppure “Avvenimenti”. Non è ammesso inserire campi nulli all’interno della tabella.

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
ID	NUMBER(3)		PRI
TIPO	VARCHAR2(30 BYTE)		

Tabella 3

C’è poi una tabella **RSS\_GOL\_CHANNEL** (**Tabella 4**) che contiene i dati relativi alle caratteristiche del feed, quali il nome del canale delle notizie, l’URL della pagina che corrisponde al canale, la descrizione, la data di pubblicazione e l’ultima data di modifica delle notizie inserite: queste informazioni, nel complesso, vengono chiamate *channel*. Durante lo sviluppo della nostra applicazione lato client non abbiamo implementato alcuni gruppi di campi (motivo per cui non sono presenti nella lista dei campi di questa tabella) pur avendolo fatto nelle due servlet che generano il codice XML: questo perché si è pensato che di questi campi non ci fosse attualmente bisogno per le necessità dell’Ente, pur creando almeno una struttura (le due servlet) già adatta a creare il giusto codice XML del feed RSS. Il primo gruppo di campi è quello di *TEXTINPUT* che specifica una richiesta di input che può essere visualizzata con il canale, utile per specificare ad esempio un eventuale campo input per un motore di ricerca o per consentire all’utente di dare un feedback, anche se molti aggregatori ignorano tale elemento durante la fase di parsing del codice XML. Un esempio di come viene reso un simile campo è dato nel seguente pezzo di codice, in cui *title* rappresenta l’etichetta sul bottone di submit, *description* è la descrizione della casella di input, *name* è il nome dell’oggetto testo nella casella di input e *link* è l’URL dello script CGI che gestisce il campo di input:

```
<textinput>
  <title>Search</title>
  <description>Search W3Schools</description>
  <name>"searchform"</name>
  <link>"http://www.google.com/search"</link>
</textinput>
```

L'altro gruppo di campi è quello di *CLOUD* che specifica un'applicazione Web che supporta l'interfaccia *rssCloud* che può essere implementata in HTTP-POST, XML-RPC oppure SOAP 1.1. Lo scopo di questa interfaccia è di permettere ai processi di registrarsi con una *cloud*, implementando un protocollo di pubblicazione-sottoscrizione piuttosto leggero per gli RSS Feed, consentendo di essere notificati degli aggiornamenti del channel. La maggior parte degli aggregatori effettua il controllo degli eventuali cambiamenti / aggiornamenti notizie una volta all'ora (se non diversamente specificato), ma in alcune situazioni si desidera che tali cambiamenti vengano immediatamente notificati. La notifica rende possibile di essere sempre aggiornati pur salvaguardando banda e cicli macchina del server che ospita il feed RSS. Un esempio dell'uso dell'elemento *cloud* è il seguente, in cui viene detto all'aggregatore di inviare un messaggio XML-RPC a *radio.xmlstoragesystem.com* sulla porta 80, con path */RPC2* alla procedura chiamata *xmlStorageSystem.rssPleaseNotify* che si incarica di effettuare la notifica all'host che si è registrato al servizio di notifica:

```
<cloud domain="radio.xmlstoragesystem.com" port="80" path="/RPC2"
registerProcedure="xmlStorageSystem.rssPleaseNotify" protocol="xml-rpc"
/>
```

Il funzionamento dell'interfaccia *rssCloud* è la seguente: una workstation chiama la *cloud* da registrare e tale procedura ha ben cinque parametri: il nome della procedura che la *cloud* dovrebbe chiamare per notificare alla workstation l'avvenuto aggiornamento, la porta TCP sulla quale la macchina remota è in ascolto, il path dove si trova la procedura di ascolto, una stringa indicante il protocollo da utilizzare (*xml-rpc* o *soap*, case-sensitive), e una lista di URL di files RSS che devono essere osservati per la notifica. Il *cloud* può determinare l'indirizzo IP del richiedente attraverso la richiesta e questo perché una workstation non può fare una chiamata di registrazione per conto di qualche altra workstation, è strettamente legata a chi fa la richiesta. Il *cloud* restituisce il valore *true* se la registrazione è andata a buon fine, *false* in caso contrario. Quando un channel a cui una workstation si è registrata viene

cambiato, cloud richiama la procedura dichiarata in fase di registrazione con un unico parametro, l'URL del channel che è stato cambiato. A questo punto la workstation viene a sapere dell'avvenuto aggiornamento e può andare a leggere il channel o notificare a sua volta altre workstations dell'aggiornamento, svuotare la sua cache, mandare una e-mail o niente di tutto ciò ma in ogni caso, a notifica avvenuta, la workstation restituisce il valore true. Per convenzione le registrazioni hanno un tempo di vita di circa venticinque ore, motivo per cui le workstations dovrebbero effettuare la registrazione ogni ventiquattro ore per ogni sottoscrizione affinché siano sempre aggiornate. E' importante notare come nota conclusiva che il protocollo / formato rssCloud non funziona su aggregatori che sono isolati da Internet mediante firewalls o NAT. E' possibile farla funzionare anche con sistemi con IP dinamico purché la workstation si registri nuovamente al servizio ad ogni riavvio (o cambio di IP). Dopo aver elencato le caratteristiche di questi due gruppi di tags, bisogna iniziare a parlare del channel: la specifica del formato RSS 2.0 impone che per descrivere il canale cui appartengono le notizie siano presenti necessariamente le voci *TITLE*, *LINK* e *DESCRIPTION*, elementi che descrivono il titolo del canale, il link principale dove si trovano le notizie sul sito Web di riferimento e la descrizione dell'argomento trattato dal canale. Una parola va spesa su cosa si intende per "titolo del canale": è il nome con cui gli utenti si riferiscono al nostro servizio di feed. Se avessimo un sito HTML che contenesse le stesse informazioni riportate nel feed RSS, allora il titolo del canale dovrebbe essere lo stesso del nostro sito. Da notare anche due campi particolari, *SKIPDAYS0* e *SKIPHOURS0* riportati all'interno della nostra tabella a quali è stato assegnato loro il valore nullo. Questi campi servono a dire agli aggregatori quando possono evitare di leggere il feed. Parleremo in modo più dettagliato di questi e di altri campi quando descriveremo le servlets che generano il codice XML.

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
TITLE	VARCHAR2(100 BYTE)		
LINK	VARCHAR2(100 BYTE)		
DESCRIPTION	VARCHAR2(4000 BYTE)		
LANGUAGE	VARCHAR2(30 BYTE)	SI	
COPYRIGHT	VARCHAR2(30 BYTE)	SI	
MANAGING_EDITOR	VARCHAR2(50 BYTE)	SI	
WEBMASTER	VARCHAR2(50 BYTE)	SI	
PUB_DATE	DATE	SI	
CATEGORY	VARCHAR2(30 BYTE)	SI	
GENERATOR	VARCHAR2(50 BYTE)	SI	

DOCS	VARCHAR2(100 BYTE)	SI	
CLOUD	VARCHAR2(100 BYTE)	SI	
TTL	VARCHAR2(5 BYTE)	SI	
IMAGE_TITLE	VARCHAR2(50 BYTE)	SI	
IMAGE_URL	VARCHAR2(100 BYTE)	SI	
IMAGE_LINK	VARCHAR2(100 BYTE)	SI	
RATING	VARCHAR2(20 BYTE)	SI	
TEXTINPUT	VARCHAR2(50 BYTE)	SI	
SKIPHOURS0	VARCHAR2(2 BYTE)	SI	
SKIPDAYS0	VARCHAR2(1 BYTE)	SI	
ID	NUMBER(5)		

Tabella 4

Ci sono infine altre due tabelle che abbiamo creato per questo primo modulo di programma. La prima è **RSS\_DIRITTI\_UTENTI (Tabella 5)** che contiene le informazioni su quali utenti possono accedere a questo primo modulo del progetto e a quale tipo di notizie essi possono accedere, sia in fase di ricerca / modifica sia quella di inserimento. La seconda è **AREA (Tabella 6)** che contiene la lista degli uffici o direzioni che possono inserire le notizie, il campo *DU\_LUNG\_FORM* che serve a limitare il testo della notizia nella form in fase di inserimento o modifica, e il campo *DU\_LUNG\_PHP* che limita la lunghezza della notizia nel file PHP.

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
ID	NUMBER		
UTENTE	VARCHAR2(50 BYTE)		
DIRITTO_TIPO_NOTIZIA	NUMBER(5)		

Tabella 5

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>	<i>Default</i>
DU_CODICE	VARCHAR2(5 BYTE)			
DU_DESCR	VARCHAR2(50 BYTE)			
DU_LUNG_FORM	NUMBER(4)			400
DU_LUNG_PHP	NUMBER(3)			300

Tabella 6

Queste cinque tabelle caratterizzano la procedura per la gestione delle notizie provenienti dai singoli uffici del Comune e quando tratteremo della procedura per la gestione degli avvenimenti / manifestazioni verranno introdotte delle ulteriori tabelle. E' giusto parlare di come sia stato possibile realizzare la procedura che verifica l'utente e gli permette di accedere alle aree di sua competenza. E' stato definito per prima cosa il nome del gruppo, l'username e la password degli utenti che potevano accedere all'application server. Volendo far girare l'applicazione con Apache Jakarta

Tomcat è necessario modificare il file **tomcat-users.xml** che si trova nella directory `<HOME_DIR>/conf` (nel nostro caso tale file si trova in `/usr/local/tomcat/jakarta-5.5.4/conf` poiché abbiamo deciso che la versione finale dell'applicazione venisse resa operativa in ambiente Linux. L'aver scelto questo sistema operativo non impedisce che il nostro progetto funzioni correttamente anche in ambiente Windows con i dovuti cambi di directory nei vari files di configurazione). Qui di seguito riportiamo un esempio di configurazione:

```
<?xml version="1.0" encoding="utf-8"?>
<tomcat-users>
  <role rolename="rss" />
  <user username="utente_rss" password="****" roles="rss" />
</tomcat-users>
```

Tomcat permette di autenticare l'utente mediante l'uso del **Realm**, un database di usernames e password che identificano gli utenti validi di una certa applicazione Web (o anche un gruppo di applicazioni), più una lista di **roles** associati a ciascun utente. I *roles* sono assimilabili ai *groups* dei sistemi operativi Unix-like. In molti ambienti, in particolare nell'ambito di applicazioni Web, è preferibile far autenticare una sola volta l'utente alle applicazioni Web disponibili su un particolare virtual host. Questo meccanismo va sotto il nome di **Single Sign-On (SSO)** e permette dunque all'utente di accedere a tutti i computers e ai sistemi dove egli ha permessi di accesso senza il bisogno di inserire molteplici password di accesso. Il SSO offre molteplici vantaggi quali:

- Riduzione del tempo impiegato dagli utenti nelle operazioni di sign-on ai domini individuali, riducendo anche la possibilità di fallimenti nelle operazioni di accesso;
- Miglioramento della sicurezza mediante il ridotto bisogno per l'utente di gestire e ricordare multipli sets di informazioni di autenticazione;
- Riduzione nel tempo impiegato e miglioramento nei tempi di risposta da parte degli amministratori di sistema nell'aggiungere e rimuovere gli utenti dal sistema o nel modificare i loro diritti di accesso;
- Miglioramento della sicurezza mediante l'avanzata abilità degli amministratori di sistema di gestire l'integrità degli account degli utenti includendo la possibilità di abilitare o disattivare i singoli accessi dell'utente a tutte le risorse del sistema in un modo co-ordinato e consistente.

Nell'approccio di autenticazione mediante Single Sign-On (**Figura 5**) il sistema deve recuperare (dal dominio primario di sign-on) tutte le informazioni di identificazione e le credenziali dell'utente necessarie a supportare l'autenticazione dell'utente per ciascuno dei domini secondari a cui l'utente potrebbe potenzialmente richiedere di interagire. L'informazione fornita dall'utente è poi usata dai servizi del Single Sign-On all'interno del dominio primario per supportare l'autenticazione dell'utente finale in ciascuno dei domini secondari con cui l'utente richiede di interagire.

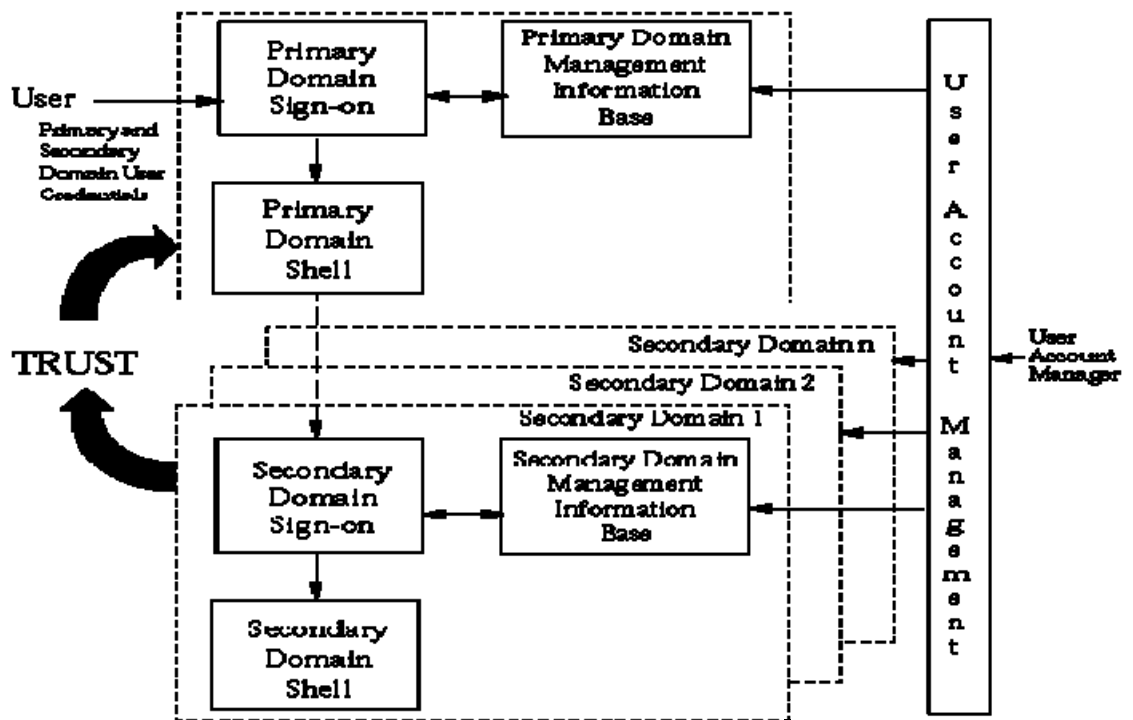


Figura 5

Una volta ultimato e messo in funzione questo programma per il Comune di Grosseto, è stato deciso di muoversi nella direzione di utilizzare il metodo di autenticazione mediante **Lightweight Directory Access Protocol (LDAP)** così da uniformare questo progetto di tesi con le loro applicazioni Web. Nel caso si voglia far uso di autenticazione mediante LDAP, si deve aggiungere le seguenti righe di configurazione nel file **server.xml** di Tomcat.

```
<Realm className="org.apache.catalina.realm.JNDIRealm" debug="99"
connectionURL="ldap://10.0.1.41:389"
userPattern="uid={0},ou=people,dc=comune,dc=grosseto,dc=it"
roleBase="ou=groups,dc=comune,dc=grosseto,dc=it"
roleName="cn"
roleSearch="(uniqueMember={0})"
/>
```

Nel proseguo della trattazione si farà sempre riferimento agli account scritti in files di configurazione e non in un server di autenticazione LDAP pur sapendo che l'applicazione finale ne fa uso.

Abbiamo accennato all'inizio di questo secondo capitolo che il nostro intento era di realizzare un'applicazione che fosse compatibile sia con prodotti commerciali sia con prodotti Open Source: volendo quindi utilizzare ad esempio Oracle OC4J 10g ciò che va fatto è analogo a quanto visto finora per Apache Jakarta Tomcat, pur cambiando il nome del file da modificare. Il file di nostro interesse si chiama **principals.xml** (o **jazn-data.xml** se vogliamo utilizzare JAZN per l'autenticazione/autorizzazione). Entrambi i files sono presenti nella directory `<HOME_DIR>/j2ee/home/config` (nelle nostre prove tali files erano presenti nella directory `/usr/local/tomcat/oc4j1013/j2ee/home/config`) benché abbiamo deciso di inserire i dati delle login nel file **principals.xml**, modificando così tale file:

```
<group name="rss">
  <description>RSS</description>
  <permission name="rmi:login" />
</group>

<user username="utente_rss" password="*****">
  <description>Utente RSS</description>
  <group-membership group="rss" />
</user>
```

E' stato poi necessario modificare anche il file **application.xml**, presente sempre nella stessa directory di configurazione dell'OC4J: questo file contiene il nome del file di login a cui va fatto riferimento. Nell'esempio da noi riportato si fa riferimento solo al file **principals.xml**, commentando il riferimento al file **jazn-data.xml**:

```
<!-- jazn provider="XML" location="./jazn-data.xml" /-->
<principals path=".principals.xml" />
```

JAZN offre una sicurezza maggiore rispetto all'utilizzo del semplice file **principals.xml** perché utilizza le password codificate, al contrario di come invece abbiamo scelto di fare noi. Una volta che sono stati definiti i gruppi e i ruoli all'interno dell'application server, è necessario modificare anche il file **web.xml**. Questo è un importante file di configurazione ed è situato nella sotto-directory **WEB-INF** di tutti i progetti Java indirizzati al Web. In questo file troviamo le sezioni

relative al tipo di autenticazione da utilizzare e ai vincoli per la sicurezza. Per il nostro progetto è stato scelto di usare la **Basic authentication** che autentica l'utente attraverso l'username e la password ottenuti da un client Web:

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <description>Tutti gli utenti che possono scrivere notizie del
  Comune</description>
  <role-name>rss</role-name>
</security-role>
```

Purtroppo questo tipo di autenticazione non è particolarmente sicura perché i dati di login vengono inviati in chiaro: di certo non è stata la scelta migliore ma abbiamo ritenuto che tale progetto non necessitasse di sicurezza maggiore, non contenendo dati sensibili. Avremmo potuto utilizzare anche la **Form-based**, in cui è possibile personalizzare la schermata di login e della pagina di errore che vengono presentate all'utente tramite un browser HTTP. Questo tipo di autenticazione, come quella da noi utilizzata, presenta il problema dell'invio in chiaro dei dati. Avessimo avuto bisogno di sicurezza, ad esempio, mandare i dati criptati mediante SSL, sarebbe stato necessario aggiungere queste righe al file web.xml, non ricorrendo a certificati che vengono utilizzati per avere la certezza dell'identità del server o del client:

```
<security-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-
    guarantee>
  </user-data-constraint>

  <web-resource-collection>
    <web-resource-name>rss</web-resource-name>
    <url-pattern>/faces/protectRSS/*</url-pattern>
    <url-pattern>/faces/menu.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>rss</role-name>
  </auth-constraint>
</security-constraint>
```

Se invece avessimo voluto mandare unicamente i dati di autenticazione in modo criptato, avremmo dovuto usare l'autenticazione di tipo **Digest** ma bisogna sottolineare che questo tipo di autenticazione ha delle ben note limitazioni, prima fra tutte l'essere niente altro che un rimpiazzo per l'autenticazione Basic. Questo vuol dire che è un sistema basato su password e (sul lato server) soffre di tutte le problematiche dei sistemi mediante password. In particolare, nessuna clausola viene fatta in questo protocollo per l'iniziale accordo sicuro tra l'utente e il server per stabilire la password dell'utente. Sia gli utenti sia gli sviluppatori di applicazioni Web dovrebbero essere consapevoli che questo protocollo non è sicuro né come **Kerberos** né come altri schemi client-side basati su chiave privata. Dall'altra parte però va detto che l'autenticazione Digest è da preferirsi al non usare nessun sistema di autenticazione ed è migliore di quanto viene comunemente utilizzato con Telnet e Ftp ma, principalmente, è migliore della Basic. Sempre all'interno del file **web.xml** dobbiamo poi indicare a quali parti del progetto si può accedere una volta autenticati e quali gruppi di utenti vi possono accedere.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>rss</web-resource-name>
    <url-pattern>/faces/protectRSS/*</url-pattern>
    <url-pattern>/faces/menu.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>rss</role-name>
  </auth-constraint>
</security-constraint>
```

Così facendo, ogni volta che qualcuno tenta di accedere agli URL indicati nel pattern, viene visualizzata una finestra di login in cui inserire l'username e la password e, una volta autenticati, si può accedere alla gestione vera e propria delle notizie. Quando tratteremo del modulo per la gestione delle notizie degli avvenimenti/manifestazioni, verranno inseriti altri pattern di indirizzi e un altro gruppo di utenti, pur mantenendo fermo il metodo attuale di autenticazione.

Cerchiamo ora di spiegare cosa succede quando l'utente si è autenticato con successo ed è pronto ad accedere alle varie notizie che gli competono: innanzi tutto viene visualizzato il menù principale mediante il quale si può accedere sia alla form per l'inserimento delle notizie sia a quella della ricerca. Sia che si scelga l'una o l'altra

opzione viene fatta una chiamata ad un metodo che si occupa di popolare l'oggetto *direzioni* che rappresenta la lista degli uffici in cui l'utente può accedere in base ai diritti che gli sono stati dati. Tale oggetto è un *ArrayList*, appartenente alla classe *java.util.ArrayList* e molto simile ad un *Vector* ma ciò che li differenzia è che quest'ultimo è una struttura dati sincronizzata mentre l'*ArrayList* non lo è: questo significa che se threads multipli accedono alla struttura e almeno uno ne varia la dimensione, allora è necessario sincronizzare la lista esternamente. Per far ciò si deve effettuare la sincronizzazione sull'oggetto che incapsula la lista ma se tale oggetto non esiste, questa deve essere racchiusa usando il metodo *Collections.synchronizedList* appartenente alla classe *java.util.Collections* e ciò avviene in fase di creazione della lista, ad esempio nel seguente modo:

```
List list = Collections.synchronizedList(new ArrayList(...));
```

La lista *direzioni* viene popolata mediante l'utilizzo di alcuni metodi quali *readTable*, *diritti* e *dirittiUtente*, tutti appartenenti alla classe da noi scritta e chiamata *it.grosseto.comune.rss.DB*. Cerchiamo di capire come operano questi metodi: *readTable* è un metodo che riceve in ingresso il nome della tabella da leggere (*AREA*), la colonna che identifica l'ID (*DU\_CODICE*), la descrizione (*DU\_DESCR*) ed una stringa che descrive i diritti associati all'utente che ha effettuato l'autenticazione. In pratica, non fa altro che fare una query sulla tabella restituendoci una lista composta dall'identificativo e dalla descrizione delle direzioni. Le condizioni di ricerca vengono generate mediante l'uso combinato di *diritti* e *dirittiUtente*. Quest'ultimo metodo permette di ottenere la lista dei codici delle direzioni dell'utente autenticato mentre il primo prende questa lista di codici e genera una stringa di ricerca che andrà passata a *readTable*.

```
direzioni=db.readTable("area","du_codice","du_descr",db.diritti(db.dirittiUtente(utenteLoggato),"du_codice"));
```

E' stata prevista la possibilità di definire un "superutente" che può gestire tutti i tipi di notizie dei vari uffici e questo semplicemente associando all'username che ci interessa, il valore "0" nel campo *DIRITTO\_TIPO\_NOTIZIA* della tabella **RSS\_DIRITTI\_UTENTI (Tabella 5)**. Va però chiarito come sia stato possibile

recuperare l'username dell'utente che ha effettuato l'autenticazione. Per far ciò, abbiamo scelto di utilizzare il metodo `getRemoteUser` della classe `javax.faces.context.ExternalContext` che consente alle API di Faces di ignorare la natura dell'ambiente che contiene l'applicazione. In particolare questa classe consente alle applicazioni basate su JavaServer Faces di essere eseguite sia in Servlet sia in Portlet (componenti web utilizzati come plug-in e gestite da un qualche container per la realizzazione e costruzione di portali Enterprise personalizzati). Una volta ottenuta l'ExternalContext è stato possibile richiamare il metodo adatto per ottenere l'username della persona che ha effettuato l'accesso all'applicazione.

```
ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
String remoteUser = externalContext.getRemoteUser();
UtenteLoggato=remoteUser;
```

### 2.3 Applicazione client per le notizie riguardanti gli avvenimenti / manifestazioni della provincia di Grosseto

Nel precedente paragrafo è stato presentato il modulo client per le notizie degli uffici del Comune: in questo verrà trattato del modulo relativo alla gestione degli avvenimenti/manifestazioni presenti nella provincia di Grosseto. In parte facciamo utilizzo delle tabelle già viste nel paragrafo precedente quale ad esempio **RSS\_GOL\_CHANNEL (Tabella 4)** che, come abbiamo già detto, descrive il canale delle notizie. Consideriamo la tabella **RSS\_GOL\_MAIN (Tabella 7)** in cui abbiamo creato un vincolo referenziale tra il campo *ID* e *PROGRE* della tabella **RSS\_AVVENIMENTI (Tabella 8)**. Quest'ultima infatti contiene le notizie immesse negli anni mediante il precedente programma di gestione ed è anche la tabella a cui accediamo quando viene fatta una ricerca da questo secondo modulo di gestione.

<i><b>Campo</b></i>	<i><b>Tipo</b></i>	<i><b>NULL</b></i>	<i><b>Chiave</b></i>
TITLE	VARCHAR2(150 BYTE)		
LINK	VARCHAR2(100 BYTE)	SI	
DESCRIPTION	VARCHAR2(4000 BYTE)		
CREATOR	VARCHAR2(8 BYTE)	SI	
CATEGORY	VARCHAR2(40 BYTE)	SI	
COMMENTS	VARCHAR2(250 BYTE)	SI	
GUID	VARCHAR2(100 BYTE)	SI	
PUBDATE	DATE	SI	
SOURCE	VARCHAR2(100 BYTE)	SI	

TIPO	NUMBER(3)		
ENCLOSURE_URL	VARCHAR2(100 BYTE)	SI	
ENCLOSURE_LENGTH	NUMBER(7)	SI	
ENCLOSURE_TYPE	VARCHAR2(20 BYTE)	SI	
ID	NUMBER(8)		PRI

Tabella 7

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
PROGRE	NUMBER(8)		PRI
DATAIN	DATE		
DATAFI	DATE		
TITOLO	VARCHAR2(150 BYTE)		
COMUNE	VARCHAR2(50 BYTE)		
ORAINS	DATE		
IPINS	VARCHAR2(18 BYTE)		
TESTO	VARCHAR2(4000 BYTE)	SI	
LOCALITA	VARCHAR2(50 BYTE)	SI	
INDIRIZZO	VARCHAR2(50 BYTE)	SI	
ORARIO	VARCHAR2(100 BYTE)	SI	
HTML	VARCHAR2(80 BYTE)	SI	
TIPO	VARCHAR2(30 BYTE)		

Tabella 8

In questo modulo le tabelle davvero fondamentali sono solamente queste due benché ce ne siano altre a cui accediamo, tipo la lista delle manifestazioni (**Tabella 9**) oppure la lista dei comuni della provincia di Grosseto (**Tabella 10**). Queste sono tabelle di lookup e le utilizziamo per popolare i menù Drop-Down all'interno delle varie form e per far questo, siamo ricorsi ad un metodo chiamato *readTable* da noi scritto, che accetta come parametri in ingresso il nome della tabella da leggere, il nome del campo che contiene la descrizione (nome delle manifestazioni o delle città) e un terzo parametro che indica la condizione che devono soddisfare i vari records. Le variabili *avv\_comuni* e *avv\_manifestazioni* a cui viene fatto riferimento nella form sono così definite e generate:

```
private List avv_comuni = new ArrayList();
private List avv_manifestazioni = new ArrayList();

avv_comuni = db.readTable("ancmoggi","comdes","comcom='53011'");
avv_manifestazioni = db.readTable("manifestazioni","tipo","");
```

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
TIPO	VARCHAR2(30)		PRI

Tabella 9

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>
COMDES	VARCHAR2(50)		
COMCOM	VARCHAR2(5)		

Tabella 10

A differenza del modulo per la gestione delle notizie comunali non abbiamo definito alcuna tabella che contiene la lista dei diritti degli utenti, demandando l'accesso all'applicazione in fase di autenticazione. Come è stato detto nel precedente paragrafo, per questa applicazione abbiamo scelto un certo tipo di autenticazione comune ad entrambi i moduli, spiegando anche quanto poco sicuro fosse questo sistema e spiegando le motivazione per cui l'abbiamo utilizzato. Per abilitare un utente ad utilizzare questa parte di applicazione, sono state aggiunte alcune righe di configurazione sia al file **tomcat-users.xml** (per Apache Jakarta Tomcat) oppure **principals.xml** (per Oracle OC4J 10g) sia al file **web.xml**. In particolare, se vogliamo abilitare all'interno di Apache Jakarta Tomcat un utente ad utilizzare sia la procedura delle notizie comunali sia quella relative agli avvenimenti, dobbiamo scrivere un file in cui abbiamo aggiunto il gruppo "avvenimenti" al pre-esistente "rss":

```
<?xml version="1.0" encoding='utf-8'?>
<tomcat-users>
  <role rolename="rss" />
  <role rolename="avvenimenti" />
  <user username="utente_completo" password="****"
    roles="rss,avvenimenti" />
</tomcat-users>
```

E logicamente va specificato nel file **web.xml** a quale parte del programma gli utenti appartenenti al gruppo "avvenimenti" sono abilitati ad accedere:

```
<security-role>
  <description>Tutti gli utenti che possono scrivere notizie di
  manifestazioni e avvenimenti</description>
  <role-name>avvenimenti</role-name>
</security-role>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>avvenimenti</web-resource-name>
    <url-pattern>/faces/protectRSSAvv/*</url-pattern>
    <url-pattern>/faces/menuavv.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
```

```
<role-name>avvenimenti</role-name>
</auth-constraint>
</security-constraint>
```

A livello di programmazione, il modulo per la gestione delle notizie comunali non è molto diverso da quello degli avvenimenti. Possiamo anzi dire che quest'ultimo non contiene la difficoltà della gestione dei molteplici uffici comunali e la relativa scrittura delle notizie nelle rispettive pagine Web poiché gli avvenimenti vengono visualizzati o mediante una speciale procedura pre-esistente utilizzata dal Comune (noi in pratica andiamo ad aggiornare l'archivio da cui questa procedura legge pur costruendo un nuovo archivio di notizie, cioè salvando nella tabella **RSS\_GOL\_MAIN (Tabella 7)** gli ultimi dati inseriti) oppure mediante gli RSS Readers (come succede anche per le notizie del Comune).

## 2.4 Servlets per la diffusione delle notizie del Comune di Grosseto e di quelle riguardanti gli avvenimenti / manifestazioni nella provincia

Prima di addentrarci nella descrizione del funzionamento della generazione del codice XML da parte delle servlets è giusto chiedersi cosa siano e perché sono da preferire ai programmi CGI o a tecnologie equivalenti. Le servlets possono essere definite come la risposta della tecnologia Java alla programmazione mediante CGI: sono programmi che sono eseguiti su un Web server, agendo come strato intermedio tra le richieste che vengono da un browser Web o altra client HTTP e i database o le applicazioni sul server HTTP. Il loro scopo è quello di:

1. **Leggere ogni informazione mandata dall'utente.** L'informazione è generalmente memorizzata da quest'ultimo all'interno di una form nella pagina Web ma potrebbe venir recuperata da una applet Java o da un programma client basato su HTTP;
2. **Cercare qualsiasi altra informazione circa la richiesta che è inserita in una richiesta HTTP.** Questa informazione include dettagli circa le capacità del browser Web, i cookies, il nome dell'host che effettua una richiesta e molte altre informazioni;
3. **Generare i risultati.** Questo processo può avvenire dialogando con un database, eseguendo una chiamata RMI o CORBA, invocando una "legacy application" cioè un'applicazione in cui una società o compagnia ha già

investito una considerevole quantità di tempo e soldi (ad esempio un database management system o DBMS che gira su mainframe o minicomputer) oppure computando direttamente il risultato;

4. **Formattare il risultato direttamente dentro il documento.** In molti casi questo implica di inserire l'informazione all'interno della pagina HTML;
5. **Settare gli appropriati parametri di risposta HTTP.** Questo significa dire al browser che tipo di documento sta per essere ritornato (ad esempio HTML, XML o altro), configurare i cookies, i parametri della cache e altri colpiti;
6. **Mandare il documento indietro al client.** Questo documento può essere mandato in formato testo (ad esempio HTML), binario (ad esempio sotto forma di immagine GIF) o anche in formato compresso tipo gzip.

Molte richieste da parte del client possono essere soddisfatte mediante la restituzione di documenti pre-costruiti e tali richieste sarebbero gestite dal server senza invocare alcuna servlet. In molti casi, comunque, un risultato statico di questo tipo non è sufficiente ed è necessario generare una specifica pagina per ogni richiesta. Ci sono molte ragioni per cui si devono generare pagine "on-fly":

- **La pagina Web è creata in base ai dati forniti dall'utente.** Un esempio è la pagina risultati di un motore di ricerca o la conferma di un ordine su un negozio on-line;
- **La pagina Web deriva da dati che cambiano frequentemente.** Per esempio il bollettino meteo o annunci di notizie;
- **La pagina Web utilizza informazioni da un database aziendale o su altre risorse lato server.** Per esempio, un sito di e-commerce potrebbe usare una servlet per costruire una pagina Web che elenca il prezzo attuale e la disponibilità di ciascun prodotto che è in vendita.

Le servlets sono più efficienti, più semplici da usare, più potenti, più portabili, più sicure e più convenienti delle tradizionali CGI e di molte altre tecnologie CGI-like:

- Con le tradizionali CGI, un nuovo processo è avviato ad ogni richiesta HTTP e se il programma è relativamente corto, l'overhead per eseguire il processo può dominare sul tempo di esecuzione. Con le servlets invece, la Java Virtual

Machine gestisce ed esegue ogni richiesta usando un thread, ben più leggero di un processo del sistema operativo. Se ci fossero ad esempio  $N$  richieste simultanee sulla stessa procedura CGI, si avrebbero  $N$  programmi CGI caricati in memoria, mentre con la servlet ci sarebbero  $N$  threads ma solo un'unica copia della servlet. In aggiunta, quando un CGI finisce di gestire la richiesta, il programma termina e questo rende difficile realizzare una cache della computazione, tenere aperta una connessione a qualche database o eseguire altre ottimizzazioni che fanno affidamento su dati persistenti. Le servlets invece rimangono in memoria anche dopo che hanno completato i risultati e questo permette di memorizzare dati tra una richiesta e l'altra;

- Le servlets sono dotate di una estensiva infrastruttura per automatizzare il parsing e la decodifica di form dati HTML, la lettura e il settaggio di headers HTTP, la gestione dei cookies, le sessioni e molte altre utilità ad alto livello. Quindi, se già si conosce il linguaggio Java risulta semplice continuare ad usarlo invece di imparare un nuovo linguaggio ad esempio il Perl o C++ anche perché il Java è un linguaggio adatto a scrivere codice affidabile e riutilizzabile;
- Le servlets supportano diverse possibilità che sono difficili o impossibili da ottenere con le normali CGI: possono infatti dialogare direttamente col Web server cosa non possibile con le CGI a meno di utilizzare delle specifiche API lato server;
- Le servlets sono scritte in linguaggio Java e seguono API standard: questo significa che se una servlet è scritta per Apache può essere eseguita senza modifiche su IBM WebSphere, Microsoft Internet Information (IIS) o altro Web server. Questo perché le servlets sono supportate direttamente o mediante qualche plug-in da virtualmente tutti i principali Web servers. Esse fanno parte della piattaforma Java 2 Enterprise Edition e il supporto industriale per le servlets sta diventando sempre più dilangante;
- Una delle principali cause di vulnerabilità all'interno dei tradizionali programmi CGI deriva dal fatto che sono spesso eseguiti da shell general-purpose del sistema operativo. Così facendo i programmatori di CGI devono stare molto attenti a filtrare certi caratteri speciali che potrebbero essere interpretati in modo speciale dalla shell. Questo è più difficile di quanto si possa pensare e le debolezze derivanti da questo problema sono

costantemente scoperte in librerie CGI largamente utilizzate. Una seconda origine di problemi è che certi programmi CGI sono processati da linguaggi che non effettuano controlli automatici sulle limitazioni degli array o delle stringhe. In C e in C++ è perfettamente accettabile allocare un array di 100 elementi e poi andare a scrivere in una posizione al di fuori di quanto è stato definito (ad esempio la 999-esima posizione). I programmatori che si dimenticano di effettuare controlli lasciano aperti i loro sistemi a deliberati o accidentali attacchi di buffer overflow. Le servlets invece non soffrono di nessuno di questi problemi e anche se una servlet esegue una chiamata remota di sistema per invocare un programma sul sistema operativo locale, ciò non avviene mediante attraverso una shell. Il controllo dei limiti di un array e altre protezioni della memoria sono una parte centrale della programmazione Java;

- Ci sono molti Web server gratuiti o molto poco costosi che sono adatti per uso personale o per siti Web che hanno un basso carico di lavoro e traffico. Fatta eccezione di Apache che è gratuito, i Web server che offrono una qualità adatta al commercio sono relativamente costosi. Tuttavia una volta in possesso di un Web server, non importa quale sia il suo costo, aggiungervi il supporto per le servlets (nel caso non sia pre-configurato per supportarle) costa veramente poco. Questo è in contrasto con molte altre alternative CGI che richiedono un significativo investimento iniziale per comprare un package proprietario.

Descriviamo ora il funzionamento delle due servlets che sono incaricate di generare il codice XML dei feed RSS: per semplicità, parleremo solo di una poiché le uniche differenze sono la tabella da cui andiamo a recuperare i dati (**RSS\_GOL** oppure **RSS\_GOL\_MAIN**) e il foglio di stile che è stato applicato sopra a ciascun codice XML. A parte queste due piccole differenze, parlare di una o dell'altra è indifferente al fine di capire come sono state realizzate. Prima di mostrare un esempio concreto di feed generato dalla nostra servlet, elenchiamo i tagss che possiamo trovare all'interno di un file RSS 2.0 standard, cioè quando non viene fatto uso di qualche schema XML aggiuntivo (**Tabella 11**):

<b><i>File Header</i></b>		
<?xml version="1.0"?>		<b>Richiesto</b>
<?xml version="1.0" encoding={charset}??>		
<rss version="2.0">		<b>Richiesto</b>
<b><i>Elementi del Channel</i></b>		
<channel>		<b>Richiesto</b>
<title>		<b>Richiesto</b>
<description>		<b>Richiesto</b>
<link>		<b>Richiesto</b>
<language>		<b>Opzionale</b>
<copyright>		<b>Opzionale</b>
<docs>		<b>Opzionale</b>
<lastBuildDate>		<b>Opzionale</b>
<managingEditor>		<b>Opzionale</b>
<pubDate>		<b>Opzionale</b>
<rating>		<b>Opzionale</b>
<skipDays>		<b>Opzionale</b>
	<day>	<b>Richiesto</b>
<skipHours>		<b>Opzionale</b>
	<hour>	<b>Richiesto</b>
<generator>		<b>Opzionale</b>
<ttl>		<b>Opzionale</b>
<image>		<b>Opzionale</b>
	<title>	<b>Richiesto</b>
	<url>	<b>Richiesto</b>
	<link>	<b>Richiesto</b>
	<description>	<b>Opzionale</b>
	<width>	<b>Opzionale</b>
	<height>	<b>Opzionale</b>
<category>		<b>Opzionale</b>
	domain	<b>Opzionale</b>
<cloud>		<b>Opzionale</b>
	domain	<b>Opzionale</b>
	port	<b>Opzionale</b>
	path	<b>Opzionale</b>
	registerProcedure	<b>Opzionale</b>
	protocol	<b>Opzionale</b>
<textinput>		<b>Opzionale</b>
	<title>	<b>Richiesto</b>
	<description>	<b>Richiesto</b>
	<name>	<b>Richiesto</b>
	<link>	<b>Richiesto</b>
<b><i>Elementi degli Item</i></b>		
<item>		<b>Richiesto</b>
<title>		<b>Richiesto</b>
<description>		<b>Richiesto</b>
<link>		<b>Opzionale</b>

<author>		<b>Opzionale</b>
<comments>		<b>Opzionale</b>
<pubDate>		<b>Opzionale</b>
<category>		<b>Opzionale</b>
	domain	<b>Opzionale</b>
<enclosure>		<b>Opzionale</b>
	url	<b>Richiesto</b>
	length	<b>Richiesto</b>
	type	<b>Richiesto</b>
<guid>		<b>Opzionale</b>
	isPermaLink	<b>Opzionale</b>
<source>		<b>Opzionale</b>
	url	<b>Richiesto</b>

Tabella 11

Come vediamo, molte voci sono opzionali e alcune hanno dei propri attributi. Cerchiamo di capire cosa succede quando la servlet viene eseguita: viene fatta una query sul campo *pubDate* (che rappresenta la data di inserimento delle notizie) della tabella **RSS\_GOL** oppure **RSS\_GOL\_MAIN** a seconda della servlet e i dati risultanti da questa operazione vengono riportati in ordine decrescente di data. Se però vengono generati troppi risultati, questi vengono filtrati e mostrati solo i primi trenta: il numero dei risultati è un parametro che è possibile definire senza dover ricompilare la servlet e nel prossimo paragrafo tratteremo in dettaglio anche di questo. Nell'esempio poco sotto riportato è possibile vedere cosa viene generato. Le prime tre righe si occupano di definire il formato, la codifica dati, il foglio di stile per la visualizzazione dei dati e anche lo schema XML utilizzato che nel nostro caso è il Dublin Core (<http://dublincore.org>). Le righe successive identificano il **channel** (o **canale**): qui troviamo il titolo, l'URL della pagina principale delle notizie, la data e l'ora di creazione del canale e quella relativa all'ultima modifica del contenuto del canale e altri. Quando qualche paragrafo fa abbiamo parlato dei campi che compongono il *channel*, abbiamo accennato dell'esistenza di due speciali campi: *SKIPDAYS0* e *SKIPHOURS0*. Dobbiamo però aggiungere che ci sono ben sette campi per gli "SKIPDAYS" (da *SKIPDAYS0* a *SKIPDAYS6*) e ben ventiquattro per gli "SKIPHOURS" (da *SKIPHOURS0* a *SKIPHOURS23*). A titolo d'esempio, ne abbiamo riportati solo due per farne capire la struttura. Cosa contengono esattamente questi due set di campi? Il primo set (SKIPDAYS) accetta come valori i numeri da 0 a 6, che rappresentano i giorni da Lunedì alla Domenica. Quando la servlet genera il codice XML del feed ed uno di questi campi è presente, viene stampato il giorno inglese corrispondente al numero immesso (0 a Monday, 1 a Tuesday, etc. etc.). Il

secondo set (SKIPHOURS) accetta invece i valori da 0 a 23 e ciascuno indica la corrispondente ora del giorno nell'arco delle ventiquattro ore. Si ricorda che anche questi due campi sono totalmente opzionali e la loro omissione non inficia la specifica del formato RSS 2.0, motivo per cui nel nostro esempio tali voci non sono presenti. Dopo questa parte riguardante il canale troviamo la parte degli **item**, cioè quella riguardante le notizie vere e proprie. Ogni blocco notizie è caratterizzato da un tag che identifica il titolo ed uno il corpo vero e proprio della notizia: secondo le specifiche questi due campi devono necessariamente essere presenti. Gli altri campi che è possibile trovare sono quelli relativi all'URL della notizia o di qualche altra risorsa ad essa correlata, alla categoria di appartenenza, all'estensore e alla data di pubblicazione o di ultima modifica della notizia. Per le tutte le informazioni sul formato delle date, ci siamo riferiti alle specifiche RFC #822 (<http://www.rfc-editor.org/rfc/rfc822.txt>) ed è stato scelto di utilizzare la notazione estesa, quella cioè in cui sono presenti le informazioni relative al giorno della settimana, al giorno, al mese scritto come lettere (ad esempio Jul per Luglio) e non come numero (ad esempio 07 riferito al caso precedente), all'anno a quattro cifre, all'ora espressa come valore sulle ventiquattro ore, ai minuti, ai secondi ed anche al fuso orario. Abbiamo poi inserito, come ultima cosa, la visualizzazione come commento di quante notizie sono state generate ma tale voce non è propria di questo formato e la sua unica utilità è quella di pura informazione, viene anche mostrato quante notizie possono essere lette, nel nostro caso sono trenta e tale valore è configurabile tramite uno speciale file di configurazione di cui parleremo in seguito.

```
<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet href="../css/rss.xsl" type="text/xsl"?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title><![CDATA[RSS GOL: Notizie del Comune di Grosseto]]></title>
  <link>http://www.comune.grosseto.it</link>
  <description><![CDATA[Informazioni riguardanti le notizie del Comune di Grosseto]]></description>
  <language>it-IT</language>
  <pubDate>Tue, 12 Jul 2005 11:59:54 +0200</pubDate>
  <lastBuildDate>Tue, 12 Jul 2005 11:59:54 +0200</lastBuildDate>
  <category><![CDATA[NEWS]]></category>
  <docs>http://backend.userland.com/rss</docs>
  <ttl>60</ttl>
  <image>
    <title>Comune di Grosseto</title>
    <url>http://www.gol.grosseto.it/pics/stemmagr\_1.gif</url>
```

```

        <link>http://www.gol.grosseto.it</link>
    </image>
    <item>
        <title><![CDATA[Concerto dell'Ensemble &agrave; cordes' du
        Conservatoire Populaire de Musique]]></title>
        <description><![CDATA[Gioved&igrave; 30 giugno alle 21 al
        Cassero Senese andranno in scena tanti giovanissimi musicisti
        dell'Ensemble &agrave; cordes' del Conservatorio popolare di
        Ginevra. L'Ensemble &egrave; composta da 14 violini, 6 viole e 5
        violoncelli e i musicisti hanno un'et&agrave; compresa tra 12 e i 19
        anni.<br>]]></description>
        <dc:creator><![CDATA[MROMLN73E25E202D]]></dc:creator>
        <category><![CDATA[Notizie Comune]]></category>
        <pubDate>Wed, 29 Jun 2005 10:12:36 +0200</pubDate>
    </item>
    <item>
        <title><![CDATA[Museo Virtuale al Cassero Senese]]></title>
        <link>http://www.gol.grosseto.it/manifesto_museo_50_70.pdf</link>
        <description><![CDATA[Gioved&igrave; 23 Giugno alle ore 18,30,
        presso il Cassero Senese, inaugurazione del Museo
        Virtuale]]></description>
        <dc:creator><![CDATA[GNNLSE48S44E202B]]></dc:creator>
        <guid
        isPermaLink="true">http://www.gol.grosseto.it/manifesto_museo_50
        _70.pdf</guid>
        <source
        url="http://www.gol.grosseto.it/manifesto_museo_50_70.pdf"/>
        <category><![CDATA[Notizie GOL]]></category>
        <pubDate>Wed, 15 Jun 2005 15:20:09 +0200</pubDate>
    </item>
<!-- Totale notizie lette: 2 [Limite: 30] -->
</channel>
</rss>

```

## 2.5 Problematiche affrontate durante la progettazione e la programmazione dell'applicazione

Dopo aver illustrato sia i moduli client che quelli server (cioè le servlets), dedicheremo questo paragrafo ad esporre più in dettaglio certi aspetti realizzativi e certe soluzioni adottate nel corso dello sviluppo del progetto. E' stato già discusso precedentemente dell'architettura MVC e delle sue tre componenti; un aspetto che accomuna le quattro parti del nostro progetto è l'utilizzo, nella parte di Model, della tecnologia **Java DataBase Connectivity (JDBC)** per accedere al database poiché tale standard si occupa di tre cose fondamentali:

1. Stabilire una connessione al database;
2. Inviare comandi SQL;
3. Processare i risultati ottenuti.

E' necessario ricordare che se si scrivono classi con l'intento di realizzare delle interrogazioni SQL ad un database, si deve necessariamente indicare quale driver JDBC deve essere utilizzato per interrogare la base di dati voluta. Solo nel caso in cui si debba utilizzare **Open DataBase Connectivity (ODBC)** di Microsoft non si deve prelevare ulteriori driver in quanto le classi necessarie al suo utilizzo sono già presenti all'interno della J2SE. Nel caso invece non ci si debba collegare tramite una fonte ODBC, si deve scaricare il driver JDBC dal sito del produttore di quello specifico database di interesse e renderlo disponibile alla classe mediante la configurazione della direttiva CLASSPATH. I driver JDBC sono divisi in quattro classi:

**Classe 1:** sono chiamati anche JDBC-ODBC Bridge e sono una implementazione per quei database che utilizzano ODBC e sono dipendenti dalla piattaforma poiché quest'ultimo fa uso delle librerie del sistema operativo;

**Classe 2:** sono le API native e sono una implementazione per i database che utilizzano librerie di accesso ai dati lato client. Il driver converte le chiamate ai metodi JDBC in chiamate all'API nativa del database. Questo genere di driver non è scritto interamente in Java perché si interfaccia con codice scritto in altri linguaggi che esegue le chiamate finali per il database. Esso è compilato per l'uso su uno specifico sistema operativo;

**Classe 3:** sono chiamati anche driver di protocollo di rete perché sono una implementazione per database che fanno uso di un middle tier tra il programma chiamante e il database vero e proprio. Questo middle tier (nello specifico è un application server) converte le chiamate ai metodi JDBC nel protocollo specifico del database. Questo tipo di classe è indipendente dalla piattaforma perché la gestione delle differenze tra le varie piattaforme è lasciata al middle tier. Inoltre, l'utilizzo di un middleware offre vantaggi in termini di sicurezza;

**Classe 4:** sono i driver del protocollo nativo e sono una implementazione di driver per database che converte le chiamate ai metodi JDBC nel protocollo nativo

del database. Questo tipo di driver, come quello di classe 3, è scritto interamente in Java e questo permette di essere indipendente dalla piattaforma. Fornisce prestazioni migliori dei driver di classe 1 e 2 perché non subisce il sovraccarico dovuto alle chiamate a ODBC o all'API nativa del database. Diversamente da loro non ha bisogno che un certo software sia installato per poter funzionare. Inoltre, siccome ogni protocollo nativo è diverso, vi sono differenti driver (generalmente forniti dal produttore) per ogni tipo di database.

I driver JDBC che abbiamo utilizzato per questo progetto (MySQL Connector/J 3.1.10 e Oracle JDBC 10.1.0.3) appartengono alla Classe 4, ciò significa che sono interamente scritti Java. Per ottenere una connessione al database abbiamo utilizzato un **datasource** ma di questo parleremo più avanti. Prima di eseguire un'applicazione di tipo enterprise dobbiamo effettuare il deployment (o distribuzione) della stessa sull'application server e per fare ciò dobbiamo impacchettare l'applicazione in un file **WAR** oppure **EAR**: questi files, infatti, contengono l'applicazione pronta per essere distribuita ed eseguita. Cerchiamo di capire più in dettaglio la composizione di tali files. Per prima cosa bisogna dire che, a seconda dell'application server, dovremo scegliere l'uno o l'altro formato. Se intendiamo utilizzare Oracle OC4J dovremo prendere in considerazione il file EAR, mentre se vogliamo utilizzare Tomcat dobbiamo necessariamente scegliere il formato WAR. Una differenza che possiamo notare è che il file EAR viene utilizzato per application server J2EE-compliant (ad esempio Oracle OC4J), ciò significa che ci sono delle caratteristiche ben precise ed uno standard a cui devono attenersi tali applicazioni. Ad esempio, nei files EAR la context root può essere definita dall'utente modificandola all'interno del file descrittore del deployment, mentre i file WAR (vedi Tomcat) non sono J2EE-compliant e come context root prendono il nome del file WAR stesso. Possiamo quindi avere più istanze dello stesso progetto semplicemente cambiando il nome del file WAR e facendone il deployment, cosa non possibile per l'altro tipo di file, a meno di non cambiare, di volta in volta, la voce relativa alla context root. Parlando di deployment bisogna ricordare che non sempre è necessario inserire in tali files tutte le librerie necessarie al corretto funzionamento delle applicazioni, infatti nel nostro elaborato in certi casi abbiamo deciso di inserire librerie comuni a più applicazioni in directory accessibili direttamente dall'application server. Nel nostro caso abbiamo utilizzato le seguenti librerie:

- Commons Beanutils 1.6.1
- Commons Collections 2.1
- Commons Digester 1.5
- Commons Logging 1.0.3
- JavaServer Pages Standard Tag Library (JSTL) 1.1
- JavaServer Faces (JSF)
- Oracle JDBC
- MySQL Connector/J

Alcune di queste non vengono inserite correttamente dagli application server nelle rispettive directory che contengono le librerie pur essendo presenti all'interno della directory /WEB-INF/lib del file WAR (o EAR). Abbiamo dunque copiato a mano queste librerie sia nella directory di librerie condivise del Tomcat (in /usr/local/tomcat/jakarta-tomcat-5.5.4/common/lib) che in quella dell'OC4J (in /usr/local/tomcat/oc4j1013/j2ee/home/lib).

Abbiamo accennato che le librerie del progetto sono contenute all'interno del file WAR / EAR, ma cosa altro contiene questo tipo di file? Sia l'uno sia l'altro sono files che possono essere compressi mediante il metodo di compressione ZIP e sono praticamente uguali fatta eccezione per la presenza, all'interno del file EAR, del file WAR e di una directory chiamata *meta-inf* contenente dei files XML di configurazione. Ci soffermeremo ad analizzare il file WAR, visto che non ci sono altre differenze con il file EAR (almeno dal punto di vista di forma). Proviamo dunque ad estrarre il contenuto del file WAR ottenendo una struttura come quella riportata:

```

/
/cea
/comgr
/css
/gol
/imgs
/infogiov
/it
/js
/mylogs
/mypackage
/newscom
/newsgol
/pariopportunita

```

*/poliziamuni*  
*/protectRSS*  
*/protectRSSAvv*  
*/pubbliaffiss*  
*/spunico*  
*/tosap*  
*/tributi*  
*/ufautsan*  
*/WEB-INF*

Abbiamo diviso in questo modo il progetto: nella directory principale (quindi in “/”) troviamo i files riguardanti il menù principale del modulo per le notizie del Comune e quello degli avvenimenti/manifestazioni, la pagina della gestione degli errori, altri files con le informazioni relative alla creazione del WAR (o EAR), il file con i messaggi dell’applicazione e, cosa importante, troviamo il file **config.properties** di cui parleremo più avanti. Nella directory */css* troviamo il foglio di stile per le pagine JSP e per le servlets, in */imgs* troviamo lo sfondo e le immagini, in */it* sono contenute tutte le classi e i backing bean del nostro progetto, in */js* troviamo il JavaScript che serve per la visualizzazione e gestione del pop-up del calendario per l’inserimento delle date nelle varie form. La directory */mypackage* contiene altre classi e backing bean, */mylogs* contiene i logs di utilizzo delle servlets mentre */protectRSS* e */protectRSSAvv* sono le directories che contengono le pagine JSP, le varie form d’inserimento e di ricerca rispettivamente del modulo per le notizie comunali e per gli avvenimenti e sono anche le directories che andremo successivamente a proteggere con l’autenticazione di tipo BASIC mentre */WEB-INF* contiene il file **web.xml** che è un importante file di configurazione, in cui abbiamo definito diversi settaggi per il corretto funzionamento dell’applicazione. Le altre directories servono come appoggio: a seconda della tipologia di notizia che viene inserita, viene scritto un file PHP chiamato **newsfile.php** all’interno della corrispondente directory e successivamente collocato all’interno delle rispettive directories di appartenenza su un altro server del Comune (sul server denominato *WWW* in **Figura 6**), differente da quello su cui è stata installata la nostra applicazione (che risiede su *AS01*), mediante l’utilizzo del comando Linux di copia remota. La scrittura di tali files su quest’altro server è stata necessaria per mantenere la compatibilità con la pre-esistente procedura di visualizzazione delle notizie (le quali scorrono in un riquadro delle varie pagine degli uffici) che accetta unicamente files in formato PHP e che, per come è stata realizzata, non può accedere direttamente al database delle notizie. Un’alternativa al

creare e successivamente copiare il file newsfile.php, è l'utilizzo di una servlet che genera on-fly le informazioni che vengono visualizzate nell'apposito riquadro delle notizie. Abbiamo accennato alla presenza di un file chiamato config.properties: in questo file, che è un file formato testo, andremo a definire dei parametri, primo tra tutti il nome **JNDI (Java Naming and Directory Interface)**. I nomi JNDI hanno un ruolo vitale nelle intranet e in Internet, in quanto permettono la condivisione di una varietà di informazioni circa gli utenti, le macchine, le reti, i servizi e le applicazioni; permettono in pratica di associare nomi ad oggetti e di recuperare oggetti in base al loro nome. Quando si lavora con applicazioni distribuite, i vari componenti non possono localizzarne altri, c'è dunque bisogno di un'altra entità che permetta ai vari componenti di localizzarsi, ed è ciò che fa il JNDI. Esso lavora su due livelli ben definiti, uno client ed uno server. Il primo si occupa dell'interfaccia tra le applicazioni e il servizio, il secondo invece si occupa della manutenzione e risoluzione delle associazioni nomi-oggetti, del controllo dell'accesso e della gestione delle operazioni eseguite sulla struttura del servizio di directory. Tornando al file di configurazione, se lo aprissimo con un qualsiasi editor di testi, vedremmo una struttura simile a questa, in cui definiamo il nome JNDI sia per Oracle sia per MySQL, più altre voci che per il momento non andremo ad analizzare.

```
# Dati riguardanti il datasource: togliere il commento all'uno
# o all'altro JndiName
# per Oracle
JndiName = jdbc/DB10gDS

# per MySQL
#JndiName = jdbc/MySqlDB

# Se true sto usando Oracle, se false uso MySQL
isOracle = true
#isOracle = false

# locazioni dei files PHP che vengono generati
tributi = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/tributi/newsfile.php
infogiov = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/infogiov/newsfile.php
pubbliaffiss = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/pubbliaffiss/newsfile.php
tosap = /usr/local/tomcat/jakarta-tomcat-5.5.4/webapps/rss/tosap/newsfile.php
uffinfo = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/uffinfo/newsfile.php
```

```

poliziamuni = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/poliziamuni/newsfile.php
uffstamp = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/uffstamp/newsfile.php
spunico = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/spunico/newsfile.php
ufautsan = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/ufautsan/newsfile.php
cea = /usr/local/tomcat/jakarta-tomcat-5.5.4/webapps/rss/cea/newsfile.php
comgr = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/comgr/newsfile.php
gol = /usr/local/tomcat/jakarta-tomcat-5.5.4/webapps/rss/gol/newsfile.php

# numero massimo di record da leggere per il file PHP
numero_max_record_php = 5

# numero massimo di feed da generare
numero_max_feed = 30

# file di configurazione per il mio Log
miolog = /usr/local/tomcat/jakarta-tomcat-
5.5.4/webapps/rss/mylogs/rsslog.properties

```

Il JDBC 2.0 ha introdotto i datasources per slegare l'applicazione dalla Factory e dall'URL di connessione, e legare il codice solo ad un semplice nome logico a cui far riferimento su JNDI: a questo nome logico JNDI deve essere associato un oggetto chiamato datasource; tale oggetto implementa l'interfaccia **javax.sql.DataSource**. I datasource sono delle Factory, termine usato per indicare delle classi in grado di creare e gestire connessioni nei confronti di generici server per le connessioni JDBC. Per utilizzare i datasources è necessario intervenire modificando, all'interno dell'application server che andremo ad usare, il suo file (o più di uno) di configurazione relativo ai datasources. Prendiamo in considerazione l'Apache Jakarta Tomcat. Nel caso di questo application server abbiamo deciso di scrivere le informazioni per accedere ai vari database nel file **context.xml** presente nella directory `<HOME_DIR>/conf` (dove `<HOME_DIR>` è la directory dove è stato installato l'application server che nel nostro caso è in `/usr/local/tomcat/jakarta-tomcat-5.5.4`) e riportando gli stessi nomi JNDI che abbiamo definito poco sopra nel file di configurazione.

```

<!-- The contents of this file will be loaded for each web application -->
<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>

```

```

<WatchedResource>META-INF/context.xml</WatchedResource>
<!-- Uncomment this to disable session persistence across Tomcat
restarts -->
<!--
<Manager pathname="" />
-->
<Resource name="jdbc/DB10gDS" auth="Container"
type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000" username="rss"
password="*****" driverClassName="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@db01.comune.grosseto.it:1521:db1"/>

<Resource name="jdbc/MySqlDB" auth="Container"
type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000" username="rss"
password="*****" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/rss?autoReconnect=true"/>
</Context>

```

Potevamo anche scegliere di mettere queste informazioni nel file **server.xml** o in **minimal-server.xml** ma ci è sembrata una soluzione più funzionale mettere tali informazioni nel file **context.xml** perché così le informazioni sono disponibili a tutte le applicazioni del contenitore e non solo ad un particolare contesto. Se prendiamo invece in considerazione Oracle OC4J, dobbiamo andare ad intervenire sul file **data-sources.xml** che si trova in `<HOME_DIR>/j2ee/home/config` per noi è in `/usr/local/tomcat/oc4j1013/j2ee/home/config`). Anche qui andremo a definire il nome JNDI e i parametri di connessione per ogni database a cui intendiamo collegarci.

```

<connection-pool name="connection-pool-RSSsuDB1">
  <connection-factory factory-
class="oracle.jdbc.pool.OracleDataSource"
user="rss"
password="*****"
url="jdbc:oracle:thin:@db01.comune.grosseto.it:1521:db1">
  </connection-factory>
</connection-pool>

<managed-data-source name="connection-RSSsuDB1"
connection-pool-name="connection-pool-RSSsuDB1"
jndi-name="jdbc/DB10gDS"/>

<connection-pool name="connection-pool-MySQL">
  <connection-factory factory-
class="oracle.jdbc.pool.OracleDataSource"
user="rss"
password="*****"
url="jdbc:mysql://localhost:3306/rss">

```

```

        </connection-factory>
    </connection-pool>

    <managed-data-source name="connection-MySQL"
        connection-pool-name="connection-pool-MySQL"
        jndi-name="jdbc/MySqlDB"/>

```

Accennando al file di configurazione **config.properties**, abbiamo solo affermato che è un file di testo che contiene parametri di configurazione ma non abbiamo detto come vi accediamo dall'interno della nostra applicazione Java. Abbiamo definito un metodo all'interno della nostra classe *it.grosseto.comune.rss.DB* che si occupa di leggerlo. E' da notare che questo file di configurazione è un *ResourceBunble* e la variabile *prop* non è altro che un oggetto della classe *java.util.ResourceBundle*. Ma che caratteristiche ha questo file? In generale un *ResourceBundle* serve a scrivere un programma che può essere facilmente tradotto in più lingue o modificato successivamente per personalizzare certi aspetti del programma come messaggi o, come nel nostro caso, parametri per il buon funzionamento dell'applicazione senza dover compilare nuovamente il progetto. Va osservato che ogni chiave del *ResourceBundle* è di tipo *String* e che è necessario convertirle nel giusto tipo a seconda dei casi.

```

public void leggiConfig() throws Exception
{
    ResourceBundle prop = ResourceBundle.getBundle("config");
    JndiName=prop.getString("JndiName");
    tributi=prop.getString("tributi");
    infogiov=prop.getString("infogiov");
    pubbliaffiss=prop.getString("pubbliaffiss");
    tosap=prop.getString("tosap");
    uffinfo=prop.getString("uffinfo");
    poliziamuni=prop.getString("poliziamuni");
    uffstamp=prop.getString("uffstamp");
    spunico=prop.getString("spunico");
    ufautsan=prop.getString("ufautsan");
    cea=prop.getString("cea");
    comgr=prop.getString("comgr");
    gol=prop.getString("gol");
    String temp="";
    temp=prop.getString("numero_max_record_php");
    numero_max_record_php=Integer.parseInt(temp);
    temp=prop.getString("isOracle");
    if (temp.equalsIgnoreCase("false")) { isOracle=false; }
    else if(temp.equalsIgnoreCase("true")) { isOracle=true; }
}

```

Il *ResourceBundle* non era l'unica possibilità per gestire i parametri di configurazione, potevamo ad esempio metterli all'interno del file **web.xml** e recuperarli dal contesto, come in questo caso di prova, con il risultato che i parametri sarebbero stati visti da tutte le classi e servlets.

```
<context-param>
  <description>questo parametro appartiene al contesto, lo vedono tutte
  le servlets</description>
  <param-name>param1</param-name>
  <param-value>prova</param-value>
</context-param>
```

Per leggere tale dato basta fare:

```
String param1 =
getServletConfig().getServletContext().getInitParameter("param1");
```

Invece per limitare la visibilità di un parametro ad una data servlet o ad una classe, bastava inserire queste impostazioni all'interno del file **web.xml** nel seguente modo:

```
<servlet>
  <servlet-name>ParametriIniziali</servlet-name>
  <servlet-class>mypackage1.ParametriIniziali</servlet-class>
  <init-param>
    <description>Questo parametro appartiene solo a questa
    servlet</description>
    <param-name>paramServlet</param-name>
    <param-value>soloPerQuestaServlet</param-value>
  </init-param>
</servlet>
```

Per recuperare tale dato bisogna fare:

```
String paramServlet = getServletConfig().getInitParameter("paramServlet");
```

Tornando al parametro JNDI, una volta che è stato recuperato tramite uno di questi modi, basterà semplicemente usare un altro metodo, anch'esso presente nella nostra classe *it.grosseto.comune.rss.DB*, per poter effettuare la connessione al database. *Context* è un oggetto appartenente alla classe *javax.naming.Context* mentre *InitialContext* fa parte di *javax.naming.InitialContext*. *InitialContext* è il contesto di partenza per eseguire le operazioni di associazione dei nomi, le quali sono relative ad

un contesto che implementa l'interfaccia Context, fornendo la base per la risoluzione dei nomi.

```
public Connection connectDB() throws Exception
{
    Context inic = new InitialContext();
    Context ctx = (Context) inic.lookup("java:comp/env");
    DataSource nativeDS = (DataSource) ctx.lookup(JndiName);
    Connection conn = nativeDS.getConnection();
    return conn;
} // connectDB
```

Come abbiamo detto, nella nostra applicazione è presente un altro importante file di configurazione, il file **web.xml** nel quale andremo ad inserire delle altre informazioni relative ai datasource come riportate in esempio:

```
<resource-ref>
    <description>Connessione a MySql</description>
    <res-ref-name>jdbc/mysqlDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>

<resource-ref>
    <description>Connessione a Oracle</description>
    <res-ref-name>jdbc/DB10gDS</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Dopo questa parte di configurazione generale, il nostro intento ora è quello di mostrare come abbiamo operato in certi punti significativi del nostro codice. Consideriamo, ad esempio, la situazione in cui, dalla pagina JSP di ricerca (**search.jsp**) del modulo di gestione delle notizie comunali (lo stesso discorso vale anche per quello degli avvenimenti: **searchavv.jsp**), vogliamo trovare dei record che abbiano i requisiti che ci interessano, risultati che poi verranno visualizzati nell'apposita pagina **result.jsp** (o **resultavv.jsp**). Il metodo che andiamo a chiamare accetta in ingresso l'identificativo della notizia, la sua data di inizio e di fine pubblicazione, la direzione dell'ufficio comunale, il tipo di notizia ed anche l'username della persona che sta eseguendo la ricerca. Questo metodo si chiama *searchByKey*, fa parte di *it.grosseto.comune.rss.DB*, e ha compito di popolare la variabile *risultati* che è un *Vector* appartenente alla classe *java.util.Vector*. Il *Vector* è come un array (vi possiamo quindi accedere tramite un indice) ma la differenza è

che può variare la sua dimensione. Una volta che la variabile dei risultati è stata popolata, viene mostrata la pagina di riepilogo di ciò che è stato trovato. Qui viene riportato il codice che si occupa di popolare la variabile e che eventualmente si occupa di mostrare, nella pagina degli errori, un messaggio che informa l'utente sul motivo per cui la ricerca non è andata a buon fine.

```
public String searchbyKeys() throws Exception
{
    tipo_notizia="1"; // Vogliamo cercare solo le notizie contrassegnate
    come "Informazioni sito"
    try
    {
        risultati =
        db.searchByKeys(id,dataStart,dataEnd,direzione,tipo_notizia,u
        tenteLoggato);
        max=risultati.size();
        return "success";
    }
    catch (Exception e)
    {
        FacesContext context = FacesContext.getCurrentInstance();
        FacesMessage message = new FacesMessage("Eccezione
        verificatasi durante la ricerca nel database. " +e.toString());
        context.addMessage("errorForm", message);
        return "failure";
    }
}
```

Una volta che il metodo *searchByKeys* restituisce il Vector riempito con qualche dato, c'è da recuperare le informazioni delle singole componenti. Esse sono formate da dati complessi in quanto ogni componente è un JavaBean che contiene le seguenti informazioni: l'ID, la data di pubblicazione, il codice dell'ufficio di appartenenza della notizia e di che tipo è. Ci troviamo dunque davanti alla situazione di dover recuperare dati da un Vector di JavaBean e lo vogliamo fare utilizzando i metodi messi a disposizione dal framework JSF. Nel frammento di codice poco sotto riportato, facente parte della pagina dei risultati (**result.jsp**), abbiamo evidenziato tre blocchi interessanti in cui abbiamo cercato di risolvere altrettante nostre esigenze. In questa pagina, come in quella relativa ai risultati della ricerca degli avvenimenti (**resultavv.jsp**) è stato fatto uso di alcune **tag libraries** messe a disposizione da JavaServer Pages. Cosa è esattamente una tag library? In JSP le **actions** sono elementi che possono creare e accedere ad oggetti del linguaggio di programmazione ed influenzare il flusso dell'output. La specifica di JSP definisce ben sei actions

standard che devono essere fornite da qualsiasi implementazione JSP-compliant. In aggiunta alle actions standard, la tecnologia JSP v1.1 supporta lo sviluppo di moduli riusabili chiamati **custom actions** che vengono invocati mediante l'utilizzo di **custom tag** all'interno delle pagine JSP: una tag library è quindi una collezione di custom tags. Alcuni esempi dei compiti che possono essere svolti dalle custom actions includono l'elaborazione di form di dati, l'accesso a database e ad altri servizi Enterprise quali e-mail e directories o gestore di flusso. Quando non erano state ancora introdotte le custom actions, i componenti JavaBeans in unione a scriptlets erano l'unico meccanismo per compiere tali compiti ma lo svantaggio di tale approccio è quello di rendere più complicata e difficile la gestione e manutenzione delle pagine JSP. Le custom actions risolvono questi problemi mediante un alto livello di programmazione a componenti poiché incapsulano tasks ricorrenti così che possano essere riutilizzati in più di un'applicazione e aumentare la produttività mediante la divisione dei compiti tra chi sviluppa e chi utilizza le librerie. Nel progetto da noi realizzato è stato fatto uso delle seguenti tags libraries:

- **JSF Core.** Permette di essere indipendenti da qualsiasi RenderKit. E' caratterizzata da tags con prefisso **f**;
- **JSF HTML.** Contiene i componenti JSF per la generazione di codice HTML. E' generalmente indicata con il prefisso **h**;
- **JSTL 1.1 Core.** Fornisce tags sostanzialmente paralleli ai costrutti strutturali della programmazione in Java, al fine di ridurre al minimo il bisogno di inserire scriptlet direttamente nelle pagine JSP. Tale libreria implementa anche alcune actions volte alla manipolazione di URL e contenuti esterni. Generalmente indicata con il prefisso **c**;

Di queste tre tag libraries, quella che abbiamo maggiormente sfruttato è stata l'ultima come vedremo tra poco. Una prima esigenza che ci ha permesso di risolvere è stata quella di riuscire a creare una tabella che avesse una riga alternativamente bianca o grigia chiara, creando un certo effetto visivo e per far questo abbiamo utilizzato il tag **<c:set>**. Tramite questo tag, ad ogni iterazione di lettura dell'elemento del Vector, andavamo ad assegnare il valore su cui, tramite **<c:choose>**, **<c:when>** e **<c:otherwise>**, è stata poi effettuata la scelta sul colore da utilizzare per la riga corrente. La seconda era quella, appunto, di accedere ai valori della nostra variabile

contenente i risultati e abbiamo deciso di farlo tramite il tag `<c:forEach>`. In questo modo abbiamo potuto iterare la lettura dei risultati della ricerca e stamparli a video all'interno della tabella. L'ultima esigenza era quella di recuperare il valore *risultato.id* che veniva letto ad ogni iterazione, facendo in modo di renderlo sempre disponibile. Tale valore sarebbe stato passato in seguito ad un altro metodo che si sarebbe occupato, tramite un bottone presente su ogni riga della tabella, di recuperare dal nostro database il record associato a tale valore che funge da identificativo. Una volta letto il record, sarebbe poi stata mostrata la form in cui avremmo potuto modificare la notizia. Per fare questo abbiamo utilizzato un tag HTML di input che passa il *risultato.id* ad una variabile *hiddenId* in modalità appunto hidden, totalmente trasparente nonché nascosto agli occhi dell'utente.

```

<table border="2">
<tr>
  <td class="b">
    <h:outputText value="#{Message.id}"/>
  </td>
  <td class="b">
    <h:outputText value="#{Message.pubdate}"/>
  </td>
  <td class="b">
    <h:outputText value="#{Message.direzione}"/>
  </td>
</tr>

<c:set var="indice" value="0"/>
<c:forEach items="{Search.risultati}" var="risultato">
  <h:form id="resultForm">
    <h:message for="resultForm"/>
    <c:set var="indice" value="{indice+1}"/>
    <c:choose>
      <c:when test="{indice % 2 == 0}">
        <tr class="grigetto">
          <c:choose>
            <c:when test="{indice % 2 == 0}">
              <tr class="grigetto">
                <td>
                  <c:out value="{risultato.id}" />
                </td>
                <td>
                  <c:out value="{risultato.pubDate}" />
                </td>
                <td>
                  <c:out value="{risultato.direzione}" />
                </td>
              </tr>
            </c:when>
            <c:otherwise>
              <tr>
                <td>
                  <c:out value="{risultato.id}" />
                </td>
                <td>
                  <c:out value="{risultato.pubDate}" />
                </td>
                <td>
                  <c:out value="{risultato.direzione}" />
                </td>
              </tr>
            </c:otherwise>
          </c:choose>
        </tr>
      </c:when>
      <c:otherwise>
        <tr>
          <td>
            <c:out value="{risultato.id}" />
          </td>
          <td>
            <c:out value="{risultato.pubDate}" />
          </td>
          <td>
            <c:out value="{risultato.direzione}" />
          </td>
        </tr>
      </c:otherwise>
    </c:choose>
  </h:form>
</c:forEach>

```

```

        </td>
        <td>
            <input type="hidden" name="hiddenId"
            value="{risultato.id}" />
            <h:commandButton id="view"
            action="{Search.view}"
            value="{Message.view_button}" />
        </td>
    </tr>
</h:form>
</c:forEach>
</table>

```

Certamente ci potevano essere altri modi per risolvere questi tre aspetti, magari l'uso di qualche altro framework tipo Struts.

Ricordando il contenuto del file **config.properties**, c'è da sottolineare che abbiamo inserito una speciale chiave che ci permette di sfruttare certe differenze tra Oracle e MySQL per le queries sugli intervalli di date. Stiamo parlando della chiave *isOracle* la quale, trattandosi di una variabile booleana, può avere solo due valori (*true* o *false*) e a seconda di questo utilizzeremo nel primo caso la sintassi di Oracle, nel secondo quella di MySQL. Come si vede guardando i frammenti di codice successivi, questi due database hanno un *operatoreData* simile, mentre hanno un *formatoData* molto diverso, pur mantenendo uguale la sintassi per il confronto sul campo di tipo data. Le variabili *operatoreData*, *formatoData* e *where* sono degli oggetti *String*, *isOracle* è di tipo *boolean* mentre *dataStartSql* è un oggetto *java.sql.Date*.

```

if (isOracle==true)
{
    operatoreData="TO_DATE";
    formatoData="yyyy-MM-dd";
}

else
{
    operatoreData="STR_TO_DATE";
    formatoData="'%Y-%m-%d'";
}

```

Per operare invece un confronto sulle date, entrambi i database accettano ad esempio questa sintassi:

```

where=where.concat("pubdate >= "+operatoreData+"("+dataStartSql+"",
"+formatoData+"");

```

Soffermiamoci ora sul pezzo significativo di codice che ci ha permesso di generare il numero progressivo da scrivere nel campo *ID* della tabella di **RSS\_GOL** (oppure **RSS\_GOL\_MAIN**), poiché tale campo rappresenta la chiave primaria. Avevamo bisogno di un modo per generare automaticamente tale valore che doveva essere univoco. A tal riguardo abbiamo deciso di usare una *sequence* ma questa soluzione se in Oracle è direttamente implementata, in MySQL non lo è ed è stato necessario creare qualcosa che ci si avvicinasse. Cosa è dunque esattamente una *sequence*? E' una tabella in cui è memorizzato il valore successivo, il precedente e l'ultimo utilizzato. Oracle mette a disposizione questo genere di struttura. La sintassi per la generazione del valore successivo da dare all'identificativo della notizia è facile, come si vede nell'esempio. Nel caso di MySQL, invece, abbiamo creato una tabella chiamata **RSS\_SEQ\_MySQL** oppure **RSS\_GOL\_MAIN\_SEQ\_MySQL** nel caso del modulo per gli avvenimenti, entrambe costituite da un unico campo non nullo che rappresenta il nostro identificativo (**Tabella 12**). Per evitare che un ID fosse associato a più notizie differenti, abbiamo provveduto a bloccare tale tabella in lettura, rendendo possibile invece la scrittura, sbloccandola unicamente quando la generazione dell'identificativo fosse avvenuta correttamente e questi passi sono stati fatti mediante l'uso dei metodi *lockTable* e *unlockTable* da noi scritti. Il bloccaggio della tabella è stato necessario per evitare appunto che si verificasse il caso in cui più persone avessero lo stesso identificativo per la notizia appena inserita nello stesso momento. Se non avessimo previsto un meccanismo di generazione univoco di identificativo, non sarebbe poi stato possibile recuperare univocamente la notizia dal suo ID. La stringa di generazione del numero successivo è molto facile in quanto si tratta di fare un update che incrementa il valore presente nella tabella e poi viene invocata una query per recuperare tale valore. In Oracle non è necessario bloccare / sbloccare le tabelle, poiché le *sequences* sono di per sé auto-bloccanti / sbloccanti.

<i>Campo</i>	<i>Tipo</i>	<i>NULL</i>	<i>Chiave</i>	<i>Default</i>
ID	int(11)			0

Tabella 12

```

Connection conn = connectDB();
long id_next;
String querySeq="";

if (isOracle==true)
{ // Oracle
    querySeq = "select RSS_SEQ.nextval id from DUAL";
}

```

```

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(querySeq);
rs.next();
id_next = rs.getLong("id");
}
else
{ // MySQL
    ResultSet rs= null;
    PreparedStatement query;
    lockTable(conn,"RSS_SEQ_MySQL");
    querySeq = "UPDATE RSS_SEQ_MySQL SET
id=LAST_INSERT_ID(id+1)";
    PreparedStatement stmt = conn.prepareStatement(querySeq);
    stmt.execute();
    query=conn.prepareStatement("select * from RSS_SEQ_MySQL;");
    rs = query.executeQuery();
    rs.next();
    id_next = rs.getLong("id");
    unlockTable(conn);
}
}

```

E' necessario precisare che affinché la tabella **RSS\_SEQ\_MySQL** e la **RSS\_GOL\_MAIN\_SEQ\_MySQL** generino il valore dell'ID, per entrambe va definito il valore iniziale mediante una semplice chiamata insert SQL; nelle sequences il valore di partenza è definito già in fase della sua creazione e quindi non è richiesta questa fase di settaggio a posteriore. Ovviamente per generare un ID con MySQL potevamo far uso di un campo auto-incrementante ma questo avrebbe complicato il codice per mantenere la compatibilità con i due differenti database.

Un altro problema che è stato affrontato riguardava la scelta del set di caratteri da utilizzare perché avevamo la necessità di poter stampare alcuni caratteri speciali quali il simbolo dell'Euro. La nostra scelta è ricaduta sul charset "**windows-1252**" che risulta essere il più compatibile; inizialmente era stato scelto di utilizzare il set "**ISO-8859-15**" ma tale scelta è risultata non ottimale perché non pienamente compatibile con alcuni caratteri da noi utilizzati. Ciò non toglie che tale set è uno di quelli che vengono maggiormente utilizzati per applicazioni Web. Il problema della scelta del set di caratteri è anche dettato da come è configurato il computer sul quale l'applicazione dovrà girare e quindi bisogna arrivare ad un compromesso tra charset disponibili ed effettiva esigenza di utilizzo di certi particolari caratteri (tipo il simbolo dell'Euro). Un'altra problematica che è stata affrontata durante lo svolgimento di questo progetto è stata, senza dubbio, reperire informazioni dalle varie documentazioni sia circa il formato RSS, sia per il modo di configurare gli

application server. Spesso infatti la documentazione era non del tutto esaustiva oppure faceva riferimento a versioni precedenti rispetto a quella da noi utilizzata (come nel caso del Tomcat): sotto questo punto di vista, i prodotti Open Source soffrono di mancanza di costanza nell'aggiornare la propria documentazione.

## 2.6 Gli RSS Readers

Nei precedenti paragrafi abbiamo visto le caratteristiche ed anche le problematiche che sono state affrontate nella realizzazione di questo progetto: finalità di questo applicativo era permettere la diffusione delle notizie del Comune di Grosseto mediante la tecnologia dei feed RSS 2.0 utilizzando il framework JSF ed è stata data un'idea del funzionamento delle servlets che si occupano di generarne i codici XML. E' stato anche affermato precedentemente che i feed RSS sono leggibili mediante degli appositi lettori denominati RSS readers o aggregatori. Di lettori ce ne sono molti, sia Open Source sia commerciali ed è difficile dire quale sia il migliore in assoluto o consigliarne uno. Essendo il formato RSS una tecnologia ancora agli inizi, non tutti i suoi tags sono pienamente supportati od utilizzati dai vari lettori. Lo scopo di questo paragrafo non è quello di fare pubblicità ad uno specifico prodotto rispetto ad un altro ma quello di descrivere brevemente i passi da seguire per configurare un generico aggregatore di notizie, anche perché la scelta alla fine si riduce ad una semplice questione di gusto personale più che di features. Nelle nostre prove abbiamo riscontrato che i lettori attualmente disponibili sono tutti validi ed affidabili e dire quale sia il migliore è impossibile: possiamo semmai dire che ci sono lettori più intuitivi di altri e per questo sembrano migliori ma alla fine tutti i lettori leggono tutti i formati RSS, qualcuno anche il formato Atom (<http://www.atomenabled.org>) e quindi l'utente finale deve affidarsi più alle proprie impressioni che a delle recensioni fatte da altri. Una volta deciso il lettore con cui ci si sente più "in sintonia", bisogna inserire gli URL dei feed e precisamente:

- <http://www.comune.grosseto.it/rss/notProtect/rssgol> nel caso siamo interessati alle ultime notizie del Comune e dei suoi vari uffici
- <http://www.comune.grosseto.it/rss/notProtect/rssgolavv> nell'ipotesi in cui vogliamo ricevere le notizie sugli avvenimenti (mostre, congressi, sagre ed altro) che hanno luogo nella provincia di Grosseto

Tra i vari tags del formato RSS abbiamo visto che ce ne sono due che dicono al lettore in quali giorni o quale ore non effettuare l'aggiornamento dati ma ne è previsto anche un altro che dice "il tempo di vita" del canale (o **TTL**), espresso in minuti, ed indica ogni quanto vengono aggiornate le notizie o anche ogni quanto si può effettuare l'aggiornamento dei dati. Nel nostro caso è stato impostato al valore di sessanta minuti ma è possibile variare tale valore agendo sulle impostazioni del lettore utilizzato anche se certi aggregatori non permettono di scendere sotto il valore del TTL che trovano nel codice XML del feed.

## Capitolo 3

### 3.1 RSS e l'interesse che sta suscitando

Nel precedente capitolo abbiamo parlato di come sia stato relativamente facile realizzare un'applicazione Web scritta in linguaggio Java facente uso del framework JSF ed operante sia con prodotti Open Source (MySQL, Apache Jakarta Tomcat) sia con prodotti professionali (Oracle 10g, Oracle OC4J) e che permettesse di diffondere notizie mediante l'uso di feed RSS 2.0. Ormai sono sempre più i siti Web che offrono il servizio di *content syndication*, indicando come il concetto di mailing list stia pian piano svanendo. Sono nate nel corso degli ultimi anni le **RSS Directories**, siti che raccolgono gli indirizzi dei feed e li catalogano secondo certi criteri (principalmente per argomento trattato) oltre a motori di ricerca di feed. Si assiste sempre più all'uscita di software o di plug-in per software già esistenti (ad esempio programmi di posta elettronica o browser Web) che permettono di svolgere il compito di aggregatore e questo dato il notevole successo che sta riscuotendo il formato RSS. Uno dei primi esempi di integrazione di feed reader all'interno di un programma stand alone è stato Mozilla Thunderbird, un client di posta elettronica gratuito e ricco di funzionalità che tra le sue funzioni permette anche di accedere ai feed RSS e ai newsgroups, senza il bisogno di dover utilizzare altri prodotti. Anche i giganti dell'informatica come Apple e Microsoft si stanno muovendo verso l'integrazione di funzionalità di lettura di RSS all'interno dei loro nuovi sistemi operativi e dei prodotti software futuri. Ciò significa che in futuro l'RSS sarà il nuovo mezzo di comunicazione con il quale ci troveremo a lavorare e che impareremo ad amare o ad odiare. Non è da escludere che qualcuno troverà il modo di fare pubblicità mediante i feed RSS visto gli interessi dei grandi produttori software per questa nuova tecnologia. Non deve neanche sorprendere che già oggi è possibile leggere i feed RSS da qualunque luogo ci si trovi, mediante l'utilizzo di dispositivi portatili (telefoni cellulari, PDA, ecc.). Ciò è dovuto anche alle potenzialità e alla compattezza dello standard Java 2 Micro Edition (J2ME) che permette di creare applicazioni adatte a sistemi embedded molto diversi tra loro.

### 3.2 Podcasting: quando RSS incontra i files audio (e non solo)

In un mondo come quello dell'informatica dominato costantemente da nuove tecnologie e soluzioni multimediali, è inevitabile trovare prodotti che si evolvono, ciò dovuto grazie alle esigenze di chi le adopera. E' questo quanto è successo quando si è iniziato ad utilizzare il tag *ENCLOSURE*. Cosa è un tag ENCLOSURE? E' un campo opzionale di RSS 2.0 (**Tabella 11**) che permette all'editore di feed di collegare un link ad un file, creando un allegato. Un esempio di uso di questo tag è il seguente, in cui notiamo la presenza di tre attributi: *url* indica l'indirizzo dove si trova l'allegato, *length* ne specifica la sua lunghezza in bytes e infine *type* serve a specificare il suo **Content Type** secondo lo standard MIME.

```
<enclosure url="http://www.scripting.com/mp3s/weatherReportSuite.mp3"
length="12216320" type="audio/mpeg" />
```

Non esiste un vincolo al genere di file che si può specificare e questo rende il formato RSS 2.0 particolarmente indicato per usi business quali ad esempio:

- **Documenti PDF.** Pensiamo agli atti di un congresso o ad una documentazione in PDF inclusa all'interno di un feed. Ciò permetterebbe a chi interessato di accedere al contenuto senza doversi scaricare pesanti allegati via e-mail;
- **Video.** E' possibile trasmettere video ad esempio di lezioni o persino dibattiti politici;
- **Audio.** Diffusioni di canzoni, talk-shows o anche editoriali;
- **Immagini.** Le agenzie immobiliari possono utilizzare il campo allegati per mostrare le foto agli acquirenti interessati. Così possono portarsi dietro cataloghi più leggeri per mostrare ai potenziali clienti solo le immagini più recenti;
- **Download.** Pensiamo al dipartimento IT di una grande azienda che deve condurre updates sui software proprietari includendo files eseguibili o compattati nel campo ENCLOSURE, permettendo agli utenti di eseguire aggiornamenti nel momento opportuno o appena venga rilasciato un nuovo aggiornamento.

Queste sono solo alcuni dei campi in cui può essere utilizzato un feed RSS 2.0 con allegati, in particolare, quando si distribuisce feed con audio si parla di **podcasting**. Questo termine indica un metodo di pubblicare files mediante Internet, consentendo agli utenti di ricevere nuovi files in modo del tutto automatico (purché il lettore RSS sia in grado di gestire il tag ENCLOSURE). E' diventato piuttosto popolare nel 2004 grazie al downloading di files audio (principalmente MP3) all'interno di dispositivi audio portatili o personal computer. La parola "podcasting" nasce dalle parole "iPod" (il popolare riproduttore di MP3 di Apple) e "broadcasting". Benché l'iPod sia probabilmente il player MP3 scelto da molti dei primi utilizzatori del podcasting, non è necessario utilizzarlo per usufruire di questo metodo di distribuzione dei contenuti. Per l'ascolto, infatti, è sufficiente disporre di un qualsiasi apparecchio (lettore MP3, telefono cellulare, PDA, ecc.) in grado di riprodurre questi files audio. Un podcast è molto simile alla sottoscrizione di un magazine audio: l'abbonato riceve regolarmente programmi audio via Internet, e può ascoltarli nelle modalità che preferisce. I podcast differiscono dalle trasmissioni audio tradizionali su Internet in due aspetti importanti. In passato, gli ascoltatori dovevano o collegarsi ad una radio on-line ad un certo orario, o dovevano effettivamente scaricare i files audio dalle pagine Web. Ottenere i podcast è più veloce e flessibile, grazie alla tecnologia dei podcast client (come iPodder o Doppler) che provvedono automaticamente, mediante l'apposito feed RSS, a scaricare l'ultima "puntata" disponibile (o tutte quelle disponibili) di quel ben preciso podcast a cui si è "abbonati". I podcast possono essere ascoltati in ogni momento perché una copia è sul computer dell'ascoltatore, o nel suo lettore MP3, e sono automaticamente recapitati agli abbonati, così che non si rende necessaria nessuna operazione attiva di downloading. Inoltre, a differenza delle Web radio in streaming, i podcast non richiedono necessariamente un collegamento ad Internet durante l'ascolto; ciò permette ai podcast la fruizione in condizioni di mobilità, prerogativa finora riservata alla radio tradizionale. Recentemente inoltre, la moda del podcasting ha raggiunto livelli di pubblico tali da indurre la stessa Apple ad inserire nel nuovo firmware dei suoi iPod, una voce del menù dedicata al podcasting gestibile direttamente dal suo software iTunes. Ha persino definito un suo namespace per la gestione delle informazioni dei podcasting chiamandolo proprio **itunes** basandolo sulle specifiche di RSS 2.0, suscitando non poche critiche sia per il nome scelto (che è appunto un suo trademark) sia per le specifiche da usare su i suoi tags.

### 3.3 Conclusioni finali

Dopo una lunga serie di discorsi siamo giunti alla conclusione. Abbiamo fatto vedere come sia stato possibile realizzare un'applicazione di gestione e distribuzione di feed RSS 2.0, è stato anche fatto vedere che questo formato sta suscitando sempre più interesse da parte di grandi case come Apple o Microsoft a tal punto da inserire un supporto diretto all'interno dei loro prossimi sistemi operativi o prodotti software di prossima uscita. E' stato poi fatto vedere come in futuro assisteremo ad un continuo uso di feed RSS 2.0 per usi anche business e questo grazie alla presenza del tag ENCLOSURE che permettono di allegare qualsiasi file ad un feed: attualmente una applicazione di content syndication con allegati è rappresentato dal podcasting in cui chiunque può mettere a disposizione di tutti una propria compilation di brani musicali o crearsi un nuovo tipo di radio su Internet. Tanto è stato il riscontro positivo che ha ottenuto questo genere di nuova forma di distribuzione di contenuti da spingere la stessa Apple ad interessarsi al fenomeno. Non va però dimenticato che i feed RSS (ma in particolare i blog) hanno avuto un grande ruolo nel far circolare notizie circa il disastro nel Sud Est Asiatico a seguito dello Tsunami o nei recenti attacchi terroristici a Londra. Gli RSS possono essere considerati come l'agenzia ANSA del nuovo millennio: se prima le notizie andavano scovate dagli utenti ora queste arrivano direttamente ai loro computer (o terminali portatili). RSS rappresenta una vera rivoluzione all'interno della rivoluzione, basti sapere che con questo strumento possiamo essere avvisati ogni volta che i nostri siti preferiti vengono aggiornati. Questo significa che i blog più importanti sono equiparabili a delle vere e proprie agenzie di informazione; un lettore RSS è infatti né più né meno come avere le notizie ANSA aggiornate in tempo reale, con il vantaggio che arrivano da più fonti e tali fonti le sceglie l'utente. Inoltre RSS permette di monitorare i siti di interesse senza dover rilasciare la propria e-mail; in questo modo il problema di spamming viene bypassato, in quanto il content syndication non passa attraverso il protocollo di posta elettronica, motivo per cui per la lettura degli RSS è richiesto l'uso di uno speciale lettore chiamato RSS reader o aggregatore. Dunque, chiunque voglia creare una propria agenzia di stampa o abbia necessità di segnalare qualcosa (una foto, un video eccezionale o anche un file) può far uso della tecnologia RSS che sicuramente nei prossimi anni sarà sempre più importante e utilizzata.

## Capitolo 4

### 4.1 Script di generazione delle tabelle per il database Oracle

```
--
-- ANCMOGGI (Table)
--
CREATE TABLE ANCMOGGI
(COMDES VARCHAR2(50 BYTE) NOT NULL,
 COMCOM VARCHAR2(5 BYTE) NOT NULL) TABLESPACE USERS
PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255 STORAGE ( INITIAL
64K MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
BUFFER_POOL DEFAULT ) LOGGING NOCACHE NOPARALLEL;
-----
--
-- AREA (Table)
--
CREATE TABLE AREA
( DU_CODICE VARCHAR2(5 BYTE) NOT NULL,
  DU_DESCR VARCHAR2(50 BYTE) NOT NULL,
  DU_LUNG_FORM NUMBER(4) NOT NULL,
  DU_LUNG_PHP NUMBER(3) NOT NULL)
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
-----
--
-- MANIFESTAZIONI (Table)
--
CREATE TABLE MANIFESTAZIONI
( TIPO VARCHAR2(30 BYTE) NOT NULL )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 24K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
--
-- MANIFE_PK (Index)
--
CREATE UNIQUE INDEX MANIFE_PK ON MANIFESTAZIONI (TIPO)
LOGGING TABLESPACE USERS PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE ( INITIAL 16K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) NOPARALLEL;
--
-- Non Foreign Key Constraints for Table MANIFESTAZIONI
--
ALTER TABLE MANIFESTAZIONI ADD ( CONSTRAINT MANIFE_PK
PRIMARY KEY (TIPO) USING INDEX TABLESPACE USERS PCTFREE 10
INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 16K MINEXTENTS 1
MAXEXTENTS 2147483645 PCTINCREASE 0 ));
-----
```

```

--
-- RSS_AVVENIMENTI (Table)
--
CREATE TABLE RSS_AVVENIMENTI
( PROGRE NUMBER(8) NOT NULL,
  DATAIN DATE NOT NULL,
  DATAFI DATE NOT NULL,
  TITOLO VARCHAR2(150 BYTE) NOT NULL,
  COMUNE VARCHAR2(50 BYTE) NOT NULL,
  ORAINS DATE NOT NULL,
  IPINS VARCHAR2(18 BYTE) NOT NULL,
  TESTO VARCHAR2(4000 BYTE),
  LOCALITA VARCHAR2(50 BYTE),
  INDIRIZZO VARCHAR2(50 BYTE),
  ORARIO VARCHAR2(100 BYTE),
  HTML VARCHAR2(80 BYTE),
  TIPO VARCHAR2(30 BYTE) NOT NULL )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 128K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
--
-- PK_AVVENIMENTI (Index)
--
CREATE UNIQUE INDEX PK_AVVENIMENTI ON RSS_AVVENIMENTI
(PROGRE) LOGGING TABLESPACE USERS PCTFREE 10 INITRANS 2
MAXTRANS 255 STORAGE ( INITIAL 1M MINEXTENTS 1 MAXEXTENTS
2147483645 PCTINCREASE 0 BUFFER_POOL DEFAULT ) NOPARALLEL;
--
-- Non Foreign Key Constraints for Table RSS_AVVENIMENTI
--
ALTER TABLE RSS_AVVENIMENTI ADD ( CONSTRAINT
PK_AVVENIMENTI PRIMARY KEY (PROGRE) USING INDEX TABLESPACE
USERS PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 1M
MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 ));
-----
--
-- RSS_DIRITTI_UTENTI (Table)
--
CREATE TABLE RSS_DIRITTI_UTENTI
( ID NUMBER,
  UTENTE VARCHAR2(50 BYTE) NOT NULL,
  DIRITTO_TIPO_NOTIZIA NUMBER(5) NOT NULL )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
--
-- PK_RSS_DIRITTI_UTENTI (Index)
--

```

```

CREATE UNIQUE INDEX PK_RSS_DIRITTI_UTENTI ON
RSS_DIRITTI_UTENTI (ID) LOGGING TABLESPACE USERS PCTFREE 10
INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 64K MINEXTENTS 1
MAXEXTENTS 2147483645 PCTINCREASE 0 BUFFER_POOL DEFAULT )
NOPARALLEL;

--
-- Non Foreign Key Constraints for Table RSS_DIRITTI_UTENTI
--
ALTER TABLE RSS_DIRITTI_UTENTI ADD ( CONSTRAINT
PK_RSS_DIRITTI_UTENTI PRIMARY KEY (ID) USING INDEX TABLESPACE
USERS PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 64K
MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 ));

-----
--
-- RSS_GOL (Table)
--
CREATE TABLE RSS_GOL
( TITLE VARCHAR2(150 BYTE) NOT NULL,
  LINK VARCHAR2(100 BYTE),
  DESCRIPTION VARCHAR2(4000 BYTE) NOT NULL,
  CREATOR VARCHAR2(18 BYTE),
  CATEGORY VARCHAR2(40 BYTE),
  COMMENTS VARCHAR2(250 BYTE),
  GUID VARCHAR2(100 BYTE),
  PUBDATE DATE,
  SOURCE VARCHAR2(100 BYTE),
  DIREZIONE NUMBER(5),
  TIPO_NOTIZIA NUMBER(3),
  ENCLOSURE_URL VARCHAR2(100 BYTE),
  ENCLOSURE_LENGTH NUMBER(7),
  ENCLOSURE_TYPE VARCHAR2(20 BYTE),
  ID NUMBER(7) )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;

--
-- PK_ID (Index)
--
CREATE UNIQUE INDEX PK_ID ON RSS_GOL (ID) LOGGING TABLESPACE
USERS PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 64K
MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
BUFFER_POOL DEFAULT ) NOPARALLEL;

--
-- Non Foreign Key Constraints for Table RSS_GOL
--
ALTER TABLE RSS_GOL ADD ( CONSTRAINT PK_ID PRIMARY KEY (ID)
USING INDEX TABLESPACE USERS PCTFREE 10 INITRANS 2 MAXTRANS
255 STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 ));

--

```

```

-- Foreign Key Constraints for Table RSS_GOL
--
ALTER TABLE RSS_GOL ADD ( CONSTRAINT
FK_RSSGOL_RSSTIPONOTIZIE FOREIGN KEY (TIPO_NOTIZIA)
REFERENCÉS RSS_TIPO_NOTIZIE (ID));
-----
--
-- RSS_GOL_CHANNEL (Table)
--
CREATE TABLE RSS_GOL_CHANNEL
( TITLE VARCHAR2(100 BYTE) NOT NULL,
LINK VARCHAR2(100 BYTE) NOT NULL,
DESCRIPTION VARCHAR2(4000 BYTE) NOT NULL,
LANGUAGE VARCHAR2(30 BYTE),
COPYRIGHT VARCHAR2(30 BYTE),
MANAGING_EDITOR VARCHAR2(50 BYTE),
WEBMASTER VARCHAR2(50 BYTE),
PUB_DATE DATE,
CATEGORY VARCHAR2(30 BYTE),
GENERATOR VARCHAR2(50 BYTE),
DOCS VARCHAR2(100 BYTE),
CLOUD VARCHAR2(100 BYTE),
TTL VARCHAR2(5 BYTE),
IMAGE_TITLE VARCHAR2(50 BYTE),
IMAGE_URL VARCHAR2(100 BYTE),
IMAGE_LINK VARCHAR2(100 BYTE),
RATING VARCHAR2(20 BYTE),
TEXTINPUT VARCHAR2(50 BYTE),
ID NUMBER(5) NOT NULL,
SKIPHOURS0 VARCHAR2(2 BYTE),
SKIPDAYS0 VARCHAR2(1 BYTE) )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
-----
--
-- RSS_GOL_MAIN (Table)
--
CREATE TABLE RSS_GOL_MAIN
( TITLE VARCHAR2(150 BYTE) NOT NULL,
LINK VARCHAR2(100 BYTE),
DESCRIPTION VARCHAR2(4000 BYTE) NOT NULL,
CREATOR VARCHAR2(40 BYTE),
CATEGORY VARCHAR2(100 BYTE),
COMMENTS VARCHAR2(250 BYTE),
GUID VARCHAR2(100 BYTE),
PUBDATE DATE,
SOURCE VARCHAR2(100 BYTE),
ENCLOSURE_URL VARCHAR2(100 BYTE),
ENCLOSURE_LENGTH NUMBER(7),

```

```

ENCLOSURE_TYPE VARCHAR2(20 BYTE),
ID NUMBER(8),
TIPO NUMBER(3) NOT NULL )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
--
-- PK_RSSGOLMAIN (Index)
--
CREATE UNIQUE INDEX PK_RSSGOLMAIN ON RSS_GOL_MAIN (ID)
LOGGING TABLESPACE USERS PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) NOPARALLEL;
--
-- Non Foreign Key Constraints for Table RSS_GOL_MAIN
--
ALTER TABLE RSS_GOL_MAIN ADD ( CONSTRAINT PK_RSSGOLMAIN
PRIMARY KEY (ID) USING INDEX TABLESPACE USERS PCTFREE 10
INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 64K MINEXTENTS 1
MAXEXTENTS 2147483645 PCTINCREASE 0 ));
--
-- Foreign Key Constraints for Table RSS_GOL_MAIN
--
ALTER TABLE RSS_GOL_MAIN ADD ( CONSTRAINT
FK_RSSGOLMAIN_RSSAVVENIMENTI FOREIGN KEY (ID) REFERENCES
RSS_AVVENIMENTI (PROGRE));
-----
--
-- RSS_GOL_MAIN_SEQ
--
CREATE SEQUENCE RSS.RSS_GOL_MAIN_SEQ
START WITH 1
MAXVALUE 1E27
MINVALUE 1
NOCYCLE
NOCACHE
NOORDER;
-----
--
-- RSS_SEQ
--
CREATE SEQUENCE RSS.RSS_SEQ
START WITH 1
MAXVALUE 1E27
MINVALUE 1
NOCYCLE
NOCACHE
NOORDER;
-----
--

```

```

-- RSS_TIPO_NOTIZIE (Table)
--
CREATE TABLE RSS_TIPO_NOTIZIE
( ID NUMBER(3),
  TIPO VARCHAR2(30 BYTE) NOT NULL )
TABLESPACE USERS PCTUSED 0 PCTFREE 10 INITRANS 1 MAXTRANS 255
STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) LOGGING NOCACHE
NOPARALLEL;
--
-- PK_RSS_TIPO_NOTIZIE (Index)
--
CREATE UNIQUE INDEX PK_RSS_TIPO_NOTIZIE ON RSS_TIPO_NOTIZIE
(ID) LOGGING TABLESPACE USERS PCTFREE 10 INITRANS 2 MAXTRANS
255 STORAGE ( INITIAL 64K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT ) NOPARALLEL;
--
-- Non Foreign Key Constraints for Table RSS_TIPO_NOTIZIE
--
ALTER TABLE RSS_TIPO_NOTIZIE ADD ( CONSTRAINT
PK_RSS_TIPO_NOTIZIE PRIMARY KEY (ID) USING INDEX TABLESPACE
USERS PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE ( INITIAL 64K
MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 ));
-----

```

## 4.2 Script di generazione delle tabelle per il database MySQL

```

--
-- CREAZIONE DATABASE
--
CREATE DATABASE rss;
-----
--
-- CREAZIONE UTENTE DATABASE CON TUTTI I DIRITTI
--
GRANT ALL ON rss.* TO rss@'%' IDENTIFIED BY '****';
GRANT ALL ON rss.* TO rss@'localhost' IDENTIFIED BY '****';
-----
--
-- ANCMOGGI (Table)
--
CREATE TABLE ancmoggi
(COMDES VARCHAR(50) NOT NULL,
 COMCOM VARCHAR(5) NOT NULL);
-----
--
-- AREA (Table)
--
CREATE TABLE area
( DU_CODICE VARCHAR(5) NOT NULL,

```

```

DU_DESCR VARCHAR(50) NOT NULL
DU_LUNG_FORM INT(4) NOT NULL,
DU_LUNG_PHP INT(3) NOT NULL);
-----
--
-- MANIFESTAZIONI (Table)
--
CREATE TABLE manifestazioni
( TIPO VARCHAR(30) NOT NULL );
--
-- MANIFE_PK (Index)
--
CREATE UNIQUE INDEX MANIFE_PK ON manifestazioni (TIPO);
--
-- Non Foreign Key Constraints for Table MANIFESTAZIONI
--
ALTER TABLE manifestazioni ADD ( CONSTRAINT MANIFE_PK PRIMARY
KEY (TIPO) );
-----
--
-- RSS_AVVENIMENTI (Table)
--
CREATE TABLE rss_avvenimenti
( PROGRE INT(8) NOT NULL,
  DATAIN DATE NOT NULL,
  DATAFI DATE NOT NULL,
  TITOLO VARCHAR(150) NOT NULL,
  COMUNE VARCHAR(50) NOT NULL,
  ORAINS TIMESTAMP NOT NULL,
  IPINS VARCHAR(18) NOT NULL,
  TESTO VARCHAR(4000),
  LOCALITA VARCHAR(50),
  INDIRIZZO VARCHAR(50),
  ORARIO VARCHAR(100),
  HTML VARCHAR(80),
  TIPO VARCHAR(30) NOT NULL );
--
-- PK_AVVENIMENTI (Index)
--
CREATE UNIQUE INDEX PK_AVVENIMENTI ON rss_avvenimenti (PROGRE);
--
-- Non Foreign Key Constraints for Table RSS_AVVENIMENTI
--
ALTER TABLE rss_avvenimenti ADD ( CONSTRAINT PK_AVVENIMENTI
PRIMARY KEY (PROGRE) );
-----
--
-- RSS_DIRITTI_UTENTI (Table)
--
CREATE TABLE rss_diritti_utenti
( ID INT,

```

```

    UTENTE VARCHAR(50) NOT NULL,
    DIRITTO_TIPO_NOTIZIA INT(5) NOT NULL );
--
-- PK_RSS_DIRITTI_UTENTI (Index)
--
CREATE UNIQUE INDEX PK_RSS_DIRITTI_UTENTI ON rss_diritti_utenti (ID);
--
-- Non Foreign Key Constraints for Table RSS_DIRITTI_UTENTI
--
ALTER TABLE rss_diritti_utenti ADD ( CONSTRAINT
PK_RSS_DIRITTI_UTENTI PRIMARY KEY (ID) );
-----
--
-- RSS_GOL (Table)
--
CREATE TABLE rss_gol
( TITLE VARCHAR(150) NOT NULL,
  LINK VARCHAR(100),
  DESCRIPTION VARCHAR(4000) NOT NULL,
  CREATOR VARCHAR(18),
  CATEGORY VARCHAR(40),
  COMMENTS VARCHAR(250),
  GUID VARCHAR(100),
  PUBDATE TIMESTAMP,
  SOURCE VARCHAR(100),
  DIREZIONE INT(5),
  TIPO_NOTIZIA INT(3),
  ENCLOSURE_URL VARCHAR(100),
  ENCLOSURE_LENGTH INT(7),
  ENCLOSURE_TYPE VARCHAR(20),
  ID INT(7) NOT NULL);
--
-- PK_ID (Index)
--
CREATE UNIQUE INDEX PK_ID ON rss_gol (ID);
--
-- Non Foreign Key Constraints for Table RSS_GOL
--
ALTER TABLE rss_gol ADD ( CONSTRAINT PK_ID PRIMARY KEY (ID) );
--
-- Foreign Key Constraints for Table RSS_GOL
--
ALTER TABLE rss_gol ADD ( CONSTRAINT FK_RSSGOL_RSSTIPONOTIZIE
FOREIGN KEY (TIPO_NOTIZIA) REFERENCES RSS_TIPO_NOTIZIE (ID));
-----
--
-- RSS_GOL_CHANNEL (Table)
--
CREATE TABLE rss_gol_channel
( TITLE VARCHAR(100) NOT NULL,
  LINK VARCHAR(100) NOT NULL,

```

```

DESCRIPTION VARCHAR(4000) NOT NULL,
LANGUAGE VARCHAR(30),
COPYRIGHT VARCHAR(30),
MANAGING_EDITOR VARCHAR(50),
WEBMASTER VARCHAR(50),
PUB_DATE TIMESTAMP,
CATEGORY VARCHAR(30),
GENERATOR VARCHAR(50),
DOCS VARCHAR(100),
CLOUD VARCHAR(100),
TTL VARCHAR(5),
IMAGE_TITLE VARCHAR(50),
IMAGE_URL VARCHAR(100),
IMAGE_LINK VARCHAR(100),
RATING VARCHAR(20),
TEXTINPUT VARCHAR(50),
ID INT(5) NOT NULL,
SKIPHOURS0 VARCHAR(2),
SKIPDAYS0 VARCHAR(1) );

```

```

-----
--
-- RSS_GOL_MAIN (Table)
--
CREATE TABLE rss_gol_main
( TITLE VARCHAR(150) NOT NULL,
  LINK VARCHAR(100),
  DESCRIPTION VARCHAR(4000) NOT NULL,
  CREATOR VARCHAR(40),
  CATEGORY VARCHAR(100),
  COMMENTS VARCHAR(250),
  GUID VARCHAR(100),
  PUBDATE TIMESTAMP,
  SOURCE VARCHAR(100),
  ENCLOSURE_URL VARCHAR(100),
  ENCLOSURE_LENGTH INT(7),
  ENCLOSURE_TYPE VARCHAR(20),
  ID INT(8),
  TIPO INT(3) NOT NULL );
--
-- PK_RSSGOLMAIN (Index)
--
CREATE UNIQUE INDEX PK_RSSGOLMAIN ON rss_gol_main (ID);
--
-- Non Foreign Key Constraints for Table RSS_GOL_MAIN
--
ALTER TABLE rss_gol_main ADD ( CONSTRAINT PK_RSSGOLMAIN
PRIMARY KEY (ID) );
--
-- Foreign Key Constraints for Table RSS_GOL_MAIN
--

```

```

ALTER TABLE rss_gol_main ADD ( CONSTRAINT
FK_RSSGOLMAIN_RSSAVVENIMENTI FOREIGN KEY (ID) REFERENCES
RSS_AVVENIMENTI (PROGRE));
-----
--
-- RSS_GOL_MAIN_SEQ_MySQL (Table)
--
CREATE TABLE rss_gol_main_seq_mysql (id INT NOT NULL);
INSERT INTO rss_gol_main_seq_mysql (id) values (0);
-----
--
-- RSS_SEQ_MySQL (Table)
--
CREATE TABLE rss_seq_mysql (id INT NOT NULL);
INSERT INTO rss_seq_mysql (id) values (0);
-----
--
-- RSS_TIPO_NOTIZIE (Table)
--
CREATE TABLE rss_tipo_notizie
( ID INT(3),
  TIPO VARCHAR(30) NOT NULL );
--
-- PK_RSS_TIPO_NOTIZIE (Index)
--
CREATE UNIQUE INDEX PK_RSS_TIPO_NOTIZIE ON rss_tipo_notizie (ID);
--
-- Non Foreign Key Constraints for Table RSS_TIPO_NOTIZIE
--
ALTER TABLE rss_tipo_notizie ADD ( CONSTRAINT PK_RSS_TIPO_NOTIZIE
PRIMARY KEY (ID) );

```

## Appendice

### A Riferimenti Web e libri di consultazione

Apple	<a href="http://www.apple.com">http://www.apple.com</a>
AtomEnabled	<a href="http://www.atomenabled.org">http://www.atomenabled.org</a>
Dublin Core	<a href="http://dublincore.org">http://dublincore.org</a>
Engineering Computer Network Homepage - ECN @ Purdue University	<a href="https://engineering.purdue.edu/ECN/">https://engineering.purdue.edu/ECN/</a>
Hex Hub Color Codes	<a href="http://www.december.com/html/">http://www.december.com/html/</a>
JDBC Data Access API	<a href="http://developers.sun.com/product/jdbc/drivers">http://developers.sun.com/product/jdbc/drivers</a>
Kerberos: The Network Authentication Protocol	<a href="http://web.mit.edu/kerberos/www/">http://web.mit.edu/kerberos/www/</a>
MySQL	<a href="http://www.mysql.com">http://www.mysql.com</a>
OpenLDAP	<a href="http://www.openldap.org">http://www.openldap.org</a>
Oracle Corporation	<a href="http://www.oracle.com">http://www.oracle.com</a>
Podcasting Tools	<a href="http://www.podcasting-tools.com">http://www.podcasting-tools.com</a>
RDF Site Summary (RSS) 1.0	<a href="http://web.resource.org/rss/1.0/spec">http://web.resource.org/rss/1.0/spec</a>
Request for Comments (RFC)	<a href="http://www.rfc-editor.org">http://www.rfc-editor.org</a>
RSS 2.0 Specification	<a href="http://blogs.law.harvard.edu/tech/rss">http://blogs.law.harvard.edu/tech/rss</a>
RSS Specifications and RSS Feeds	<a href="http://www.rss-specifications.com/">http://www.rss-specifications.com/</a>
RSS Version Comparison	<a href="http://www.rssdotnet.com/documents/version_comparison.html">http://www.rssdotnet.com/documents/version_comparison.html</a>
Servlets and JavaServer Pages (JSP) 1.0: A Tutorial	<a href="http://www.apl.jhu.edu/%7Eehall/java/Servlet-Tutorial/">http://www.apl.jhu.edu/%7Eehall/java/Servlet-Tutorial/</a>
Sun Developer Network	<a href="http://java.sun.com">http://java.sun.com</a>
The Apache Jakarta Project	<a href="http://jakarta.apache.org">http://jakarta.apache.org</a>
Unicode	<a href="http://www.unicode.org">http://www.unicode.org</a>
UserLand RSS Central	<a href="http://rss.userland.com">http://rss.userland.com</a>
W3Schools Online Web Tutorials	<a href="http://www.w3schools.com">http://www.w3schools.com</a>

Wikipedia	<a href="http://www.wikipedia.org">http://www.wikipedia.org</a>
World Wide Web Consortium	<a href="http://www.w3.org">http://www.w3.org</a>
XML.com: What is RSS?	<a href="http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html">http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html</a>

Ben Hammersley, <i>Content Syndication with RSS</i> , Ed. O'Reilly, 2003
Heiner Stuckenschmidt, Frank van Harmelen, <i>Information Sharing on the Semantic Web</i> , Ed. Springer, 2003
Kito D. Mann, <i>JavaServer Faces in Action</i> , Ed. Manning Publications Co., 2004

## B Note finali

Sono state scritte le documentazioni, sia in formato PDF sia in HTML, relative al funzionamento della procedura di inserimento delle notizie (sia comunali sia della provincia relative agli avvenimenti). E' stata anche scritta una guida che illustra come configurare i principali lettori RSS per leggere i feed generati dalle due servlets. Per mettere al corrente gli utenti del Comune di Grosseto è stata poi scritta una e-mail indicante la possibilità di leggere le notizie mediante l'uso di feed RSS. In aggiunta a tutto questo, è stato poi realizzato un articolo di questa tesi che è stato pubblicato sul portale delle tecnologia informatiche Tecnoteca (<http://www.tecnoteca.it/howto/rss>). Per concludere questa trattazione, in **Figura 6** è riportato lo schema che illustra (in modo semplificato) la topologia della rete di calcolatori del Comune di Grosseto coinvolti nel funzionamento di questa procedura.

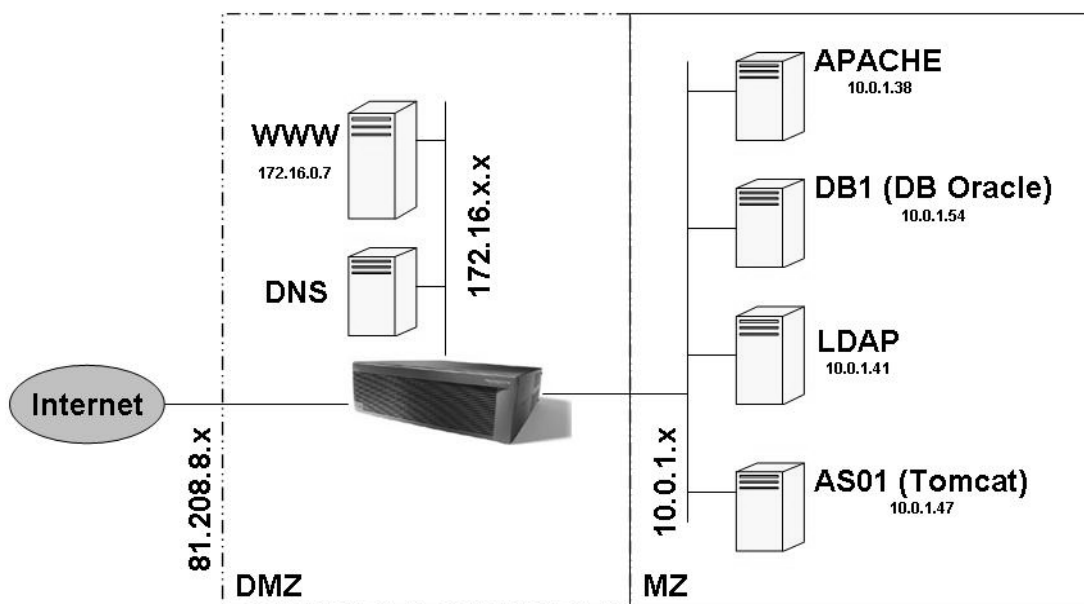


Figura 6



Modello di ponte elevabile dal *De re militari* di Vegetio, Erfurt 1512-'13