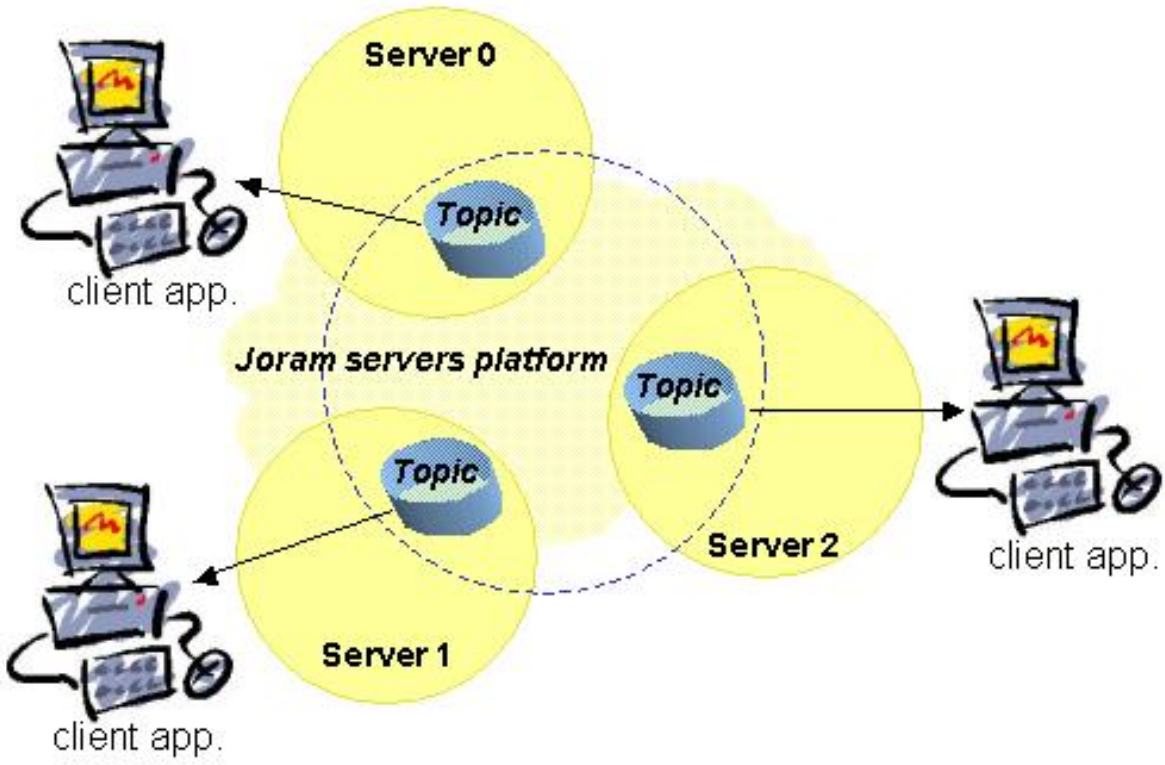
Università degli Studi di Firenze

UNIVERSITÀ DEGLI STUDI DI FIRENZE
FACOLTA DI INGEGNERIA – DIPARTIMENTO DI SISTEMI E INFORMATICA

Elaborato di Tecnologie del Software per Internet

JMSWEB 2



SISTEMA PER LO SCAMBIO DI MESSAGGI TRA APPLICAZIONI WEB

Autori:
- Bandini Roberto
- Ercoli Simone

ANNO ACCADEMICO 2005-2006

Prefazione

Il sistema descritto è un elaborato realizzato per applicare i concetti teorici descritti nel corso di Tecnologie del Software per Internet insieme ad altri corsi come ad esempio Basi di Dati.
Questo lavoro mostra un esempio concreto ed utile a mettere in pratica il sapere acquisito a conclusione delle lezioni inerenti al modulo professionalizzante “Programmatore per Applicazioni Web”.

L’elaborato da una visione trasversale dei vari argomenti trattati e della loro applicazione in un contesto reale.
Vengono usati diversi strumenti e nozioni adoperati anche nel normale contesto aziendale: JSP, Servlet, JMS, JDBC, HTML, MySQL.

Indice

1 Contesto

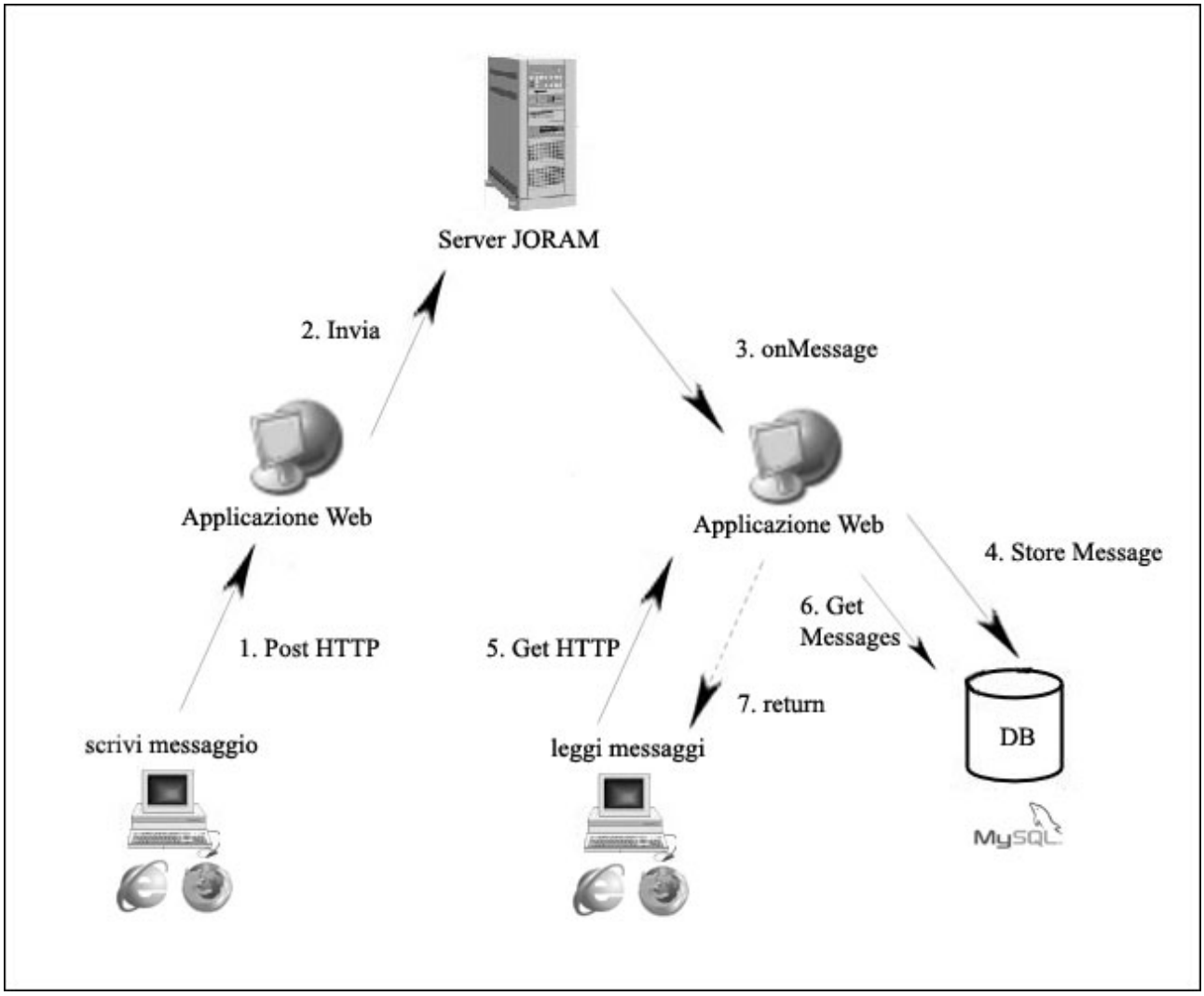
1.1 OBIETTIVO

L'obiettivo è la realizzazione di un sistema per lo scambio di messaggi tra applicazioni web.
Il sistema viene realizzato tramite la progettazione e realizzazione di una applicazione web specifica che può comunicare con n applicazioni identiche.
La comunicazione avviene tramite lo scambio di messaggi gestiti da un apposito server (Joram).

1.2 TECNOLOGIE E STRUMENTI

L'applicazione web viene realizzata tramite HTML, JSP, JAVA, SERVLET, JDBC e JMS.
Il web server usato è TomCat, mentre per la persistenza dei dati viene usato il DBMS MySQL.
Il server per la gestione dei messaggi è il JORAM.
L'ambiente di sviluppo usato è l'IDE ECLIPSE.

1.3 PRINCIPI DI FUNZIONAMENTO



Il sistema è composto da due o più applicazioni web identiche e da un server che gestisce i messaggi.
Ogni applicazione web tramite un HTML form può prendere in input dei messaggi che tramite delle servlet e oggetti Java appositi richiamano le librerie JMS per inviare i messaggi al server JORAM.

Il server, una volta ricevuto un messaggio da una applicazione, lo invia a sua volta a tutte le applicazioni ad esso collegate e registrate a un Topic ad esse comune, ovvero al quale sono sottoscritte.
Ogni applicazione che si è sottoscritta a un determinato Topic resta in ascolto dei messaggi in arrivo che memorizza automaticamente nel suo database.
L'utente tramite l'interfaccia web può vedere la lista dei messaggi arrivati, leggerli ed eliminarli.
Tramite un apposito filtro si evita che il messaggio rispedito dal server venga inoltrato anche al mittente.

2 Requisiti

In questo capitolo riportiamo i requisiti del sistema da noi progettato ovvero le funzioni, le proprietà e i vincoli che lo caratterizzano rispetto all'obiettivo per il quale è stato ideato e realizzato.

2.1 REQUISITI FUNZIONALI

1) Il sistema deve permettere l'invio di un messaggio dalla propria l'applicazione web verso le altre usando come tramite il server per la gestione dei messaggi.
2) Il sistema prevede una interfaccia web dalla quale l'utente può eseguire le varie operazioni.
3) L'applicazione prevede una pagina tramite la quale inviare un messaggio.
4) Il server Joram riceve i messaggi dai client e li inoltra a tutti i sottoscrittori tranne il mittente.
5) L'applicazione web ha un listener JMS di messaggi.
   Quando un messaggio è ricevuto il listener lo memorizza sul database tramite il DAO.
6) Tramite una pagina si può vedere la lista in ordine cronologico dei messaggi presenti sul database dell'applicazione usata.
7) L'utente può leggere i messaggi presenti nel database dell'applicazione che sta usando.
8) L'utente può cancellare i messaggi presenti sul database dell'applicazione che usa.

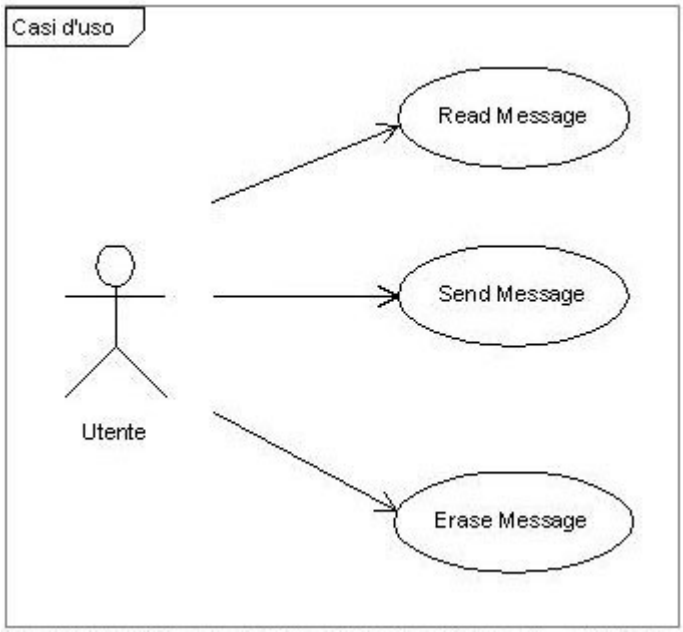
2.2 REQUISITI NON FUNZIONALI

1) Il sistema deve essere implementato tramite il linguaggio Java e la sua relativa piattaforma.
2) Il sistema deve far uso di un server Joram per quanto riguarda la gestione dei messaggi.
3) Il sistema deve poter memorizzare i messaggi su una base dati personale.

2.3 VINCOLI

1) All'avvio della applicazione web il sistema deve creare e avviare: il publisher, il subscriber con relativo message listener e il dao.
2) Il tipo di sottoscrizione verso il server di scambio messaggi deve essere di tipo durevole. Questo fa in modo che se l'applicazione web interrompe per qualche motivo la sua comunicazione col server, una volta ristabilito il funzionamento essa può ricevere i messaggi che erano in attesa tramite il fatto che il suo identificatore è sempre lo stesso.

3 UML Use Cases

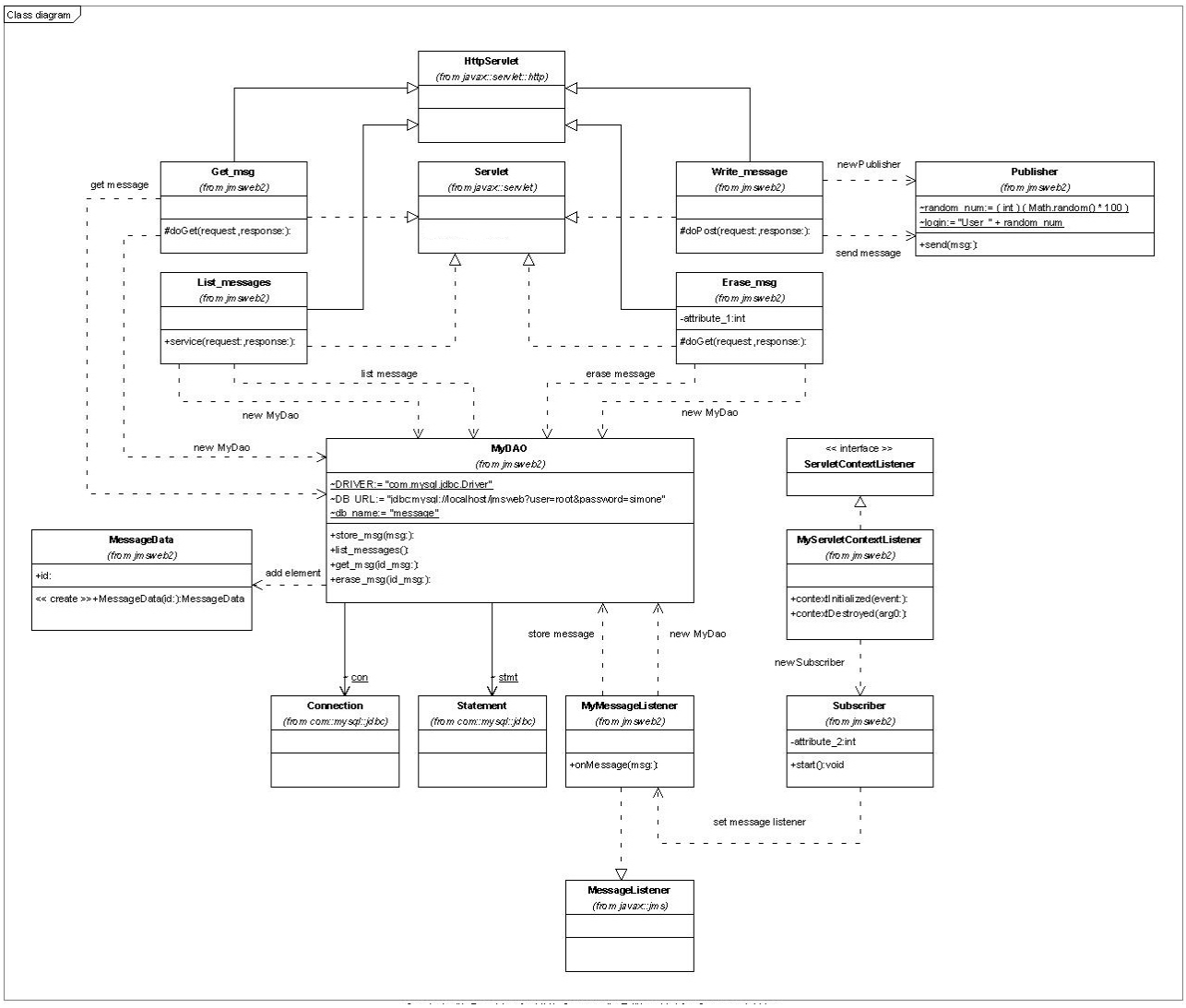


Come si può vedere dallo schema sopra riportato i casi d'uso generali del sistema da parte dell'utente sono 3.

1) READ MESSAGE: l'utente legge un messaggio dopo averlo scelto dalla lista che gli viene presentata tramite l'interfaccia web.
2) SEND MESSAGE: tramite un apposito modulo l'utente scrive e invia un messaggio che viene poi ricevuto dal server e inoltrato ai sottoscrittori.
3) ERASE MESSAGE: l'utente cancella un messaggio dopo averlo scelto dalla lista dei messaggi presenti sul database.

4 UML Class Diagram

4.1 APPLICATION CLASS DIAGRAM

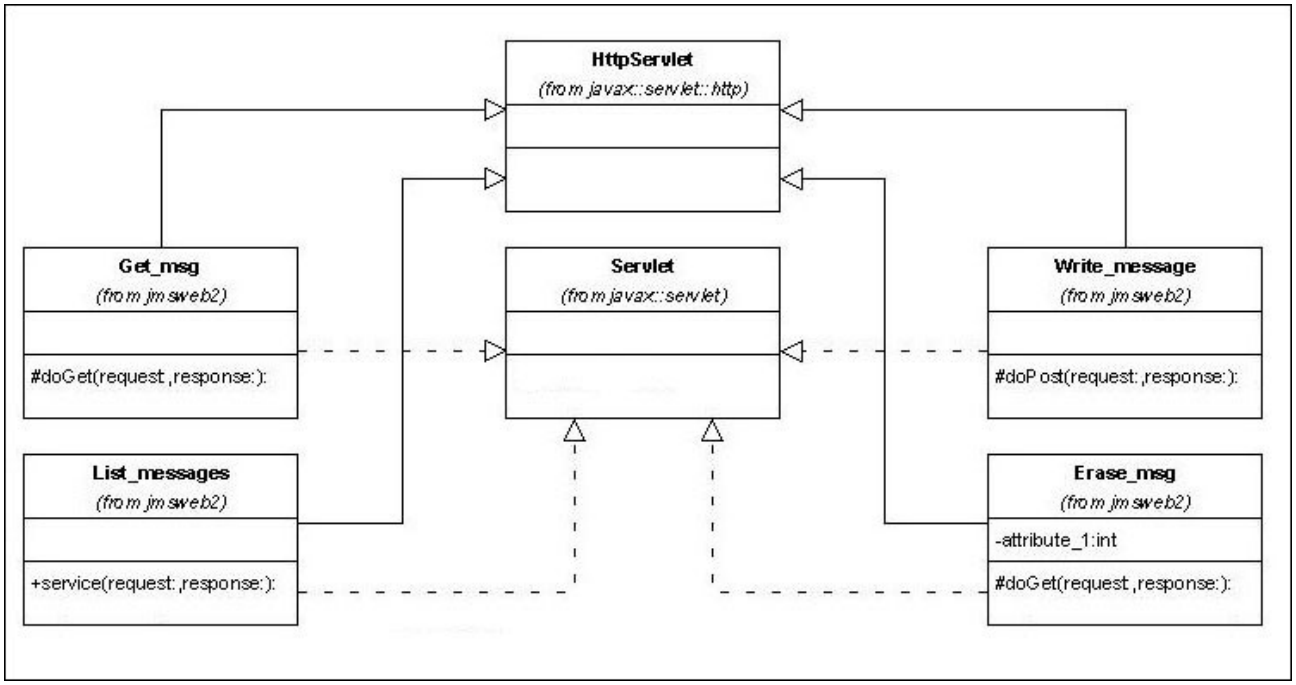


Lo schema sopra riportato è il Class Diagram generale della nostra applicazione web che comprende in linea di massima: Servlet, DAO, Publisher e Subscriber.
Questo schema ci fa capire i vari collegamenti fra le singole classi e interfacce con i loro attributi e i loro metodi.
Adesso andremo a vedere nel dettaglio questo schema suddividendolo in 3 blocchi analizzando le singole classi che li compongono.

Le servlet sono gli elementi di controllo richiamati tramite l'interfaccia web che a loro volta utilizzano gli altri oggetti come ad esempio il DAO, per effettuare le varie operazioni sul database.

Il Publisher e il Subscriber sono le classi che servono per pubblicare i messaggi sul Topic presente sul Joram e per sottoscriversi.

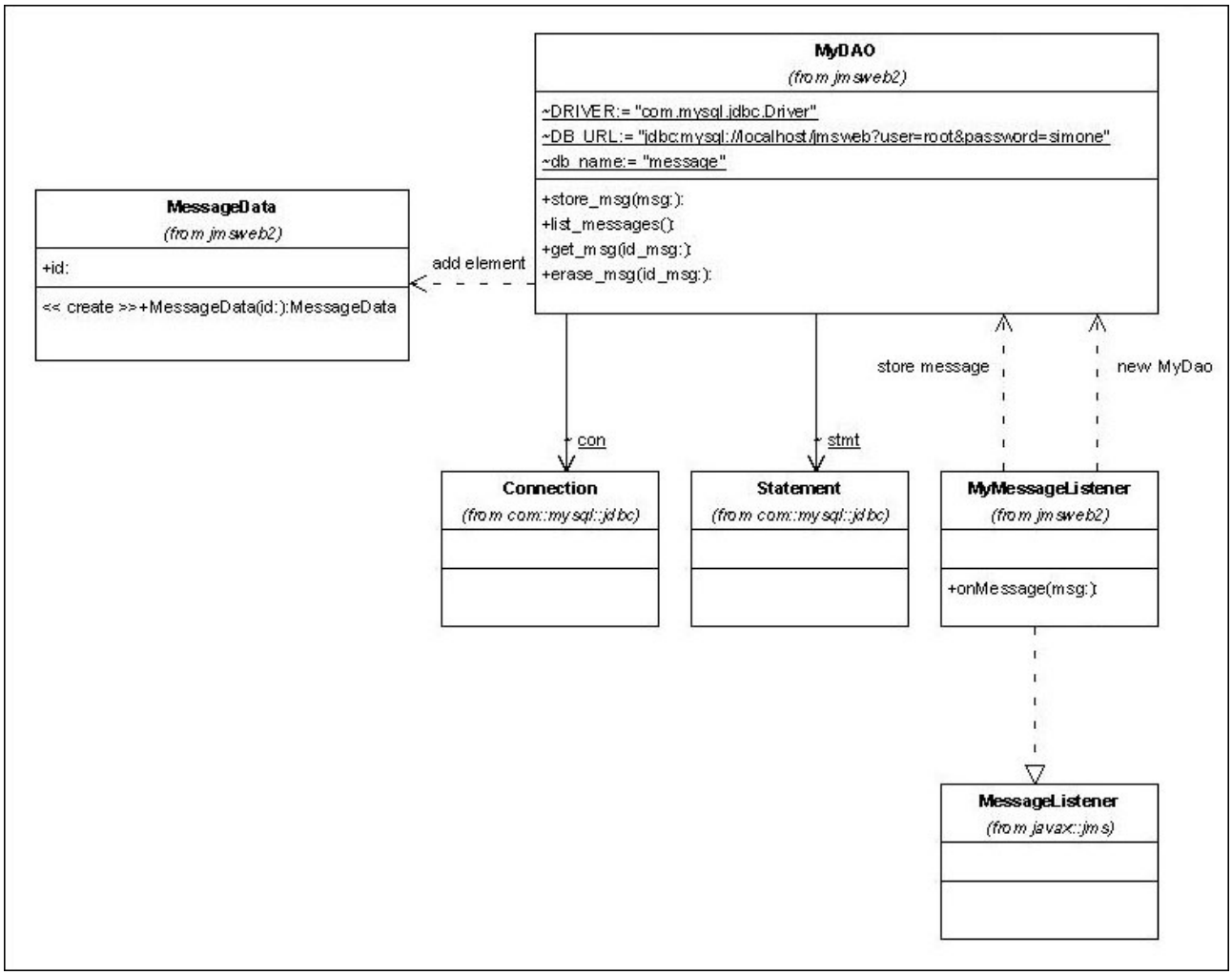
4.2 SERVLET CLASS DIAGRAM



Nello schema sopra riportato vediamo nel dettaglio la parte del class diagram che interessa le Servlet.
Come si può notare dalla figura le servlet che usiamo sono 4:

- WRITE_MESSAGE: ha un metodo doPost che serve a utilizzare i dati provenienti dalla pagina contenente il modulo, ovvero write_message.jsp.
  La servlet crea un oggetto publisher e richiama su esso un metodo send per inviare il messaggio.
  Tramite questa serie di operazioni il messaggio viene inviato al server Joram e una volta ultimato questo processo la servlet ci mostra la pagina message_sent.jsp per indicarci che il nostro messaggio è stato spedito e che possiamo procedere con un'altra operazione.
- LIST_MESSAGES: crea un oggetto DAO e applica su esso il metodo "list_messages" che restituisce tramite una collection la lista dei messaggi.
  A questo punto la servlet riporta il controllo sulla pagina message_list.jsp che ci mostra la lista vera e propria dei messaggi con ID in ordine cronologico e link per leggere e cancellare il singolo messaggio.
- GET_MSG: la servlet viene richiamata passandogli l'id del messaggio scegliendo un messaggio da leggere dalla lista.
  Viene creato un oggetto DAO sul quale si richiama il metodo get_msg passando l'id del messaggio richiesto.
  Il DAO restituisce un oggetto MessageData, contenente la stringa corrispondente al messaggio vero e proprio che viene passata dalla servlet alla pagina message.jsp.
- ERASE_MESSAGE: questa servlet funziona allo stesso identico modo della servlet GET_MSG, richiama però un altro metodo sul DAO che è erase_msg a cui viene passato l'id del messaggio per cancellarlo.
  Una volta cancellato il messaggio la servlet restituisce il controllo mostrando la pagina message_list.jsp dove ci si può accorgere che il messaggio cancellato non è per l'appunto più presente.

4. 3 DAO CLASS DIAGRAM



Il class diagram sopra riportato mostra il DAO, la classe MessageData e il MyMessage Listener.

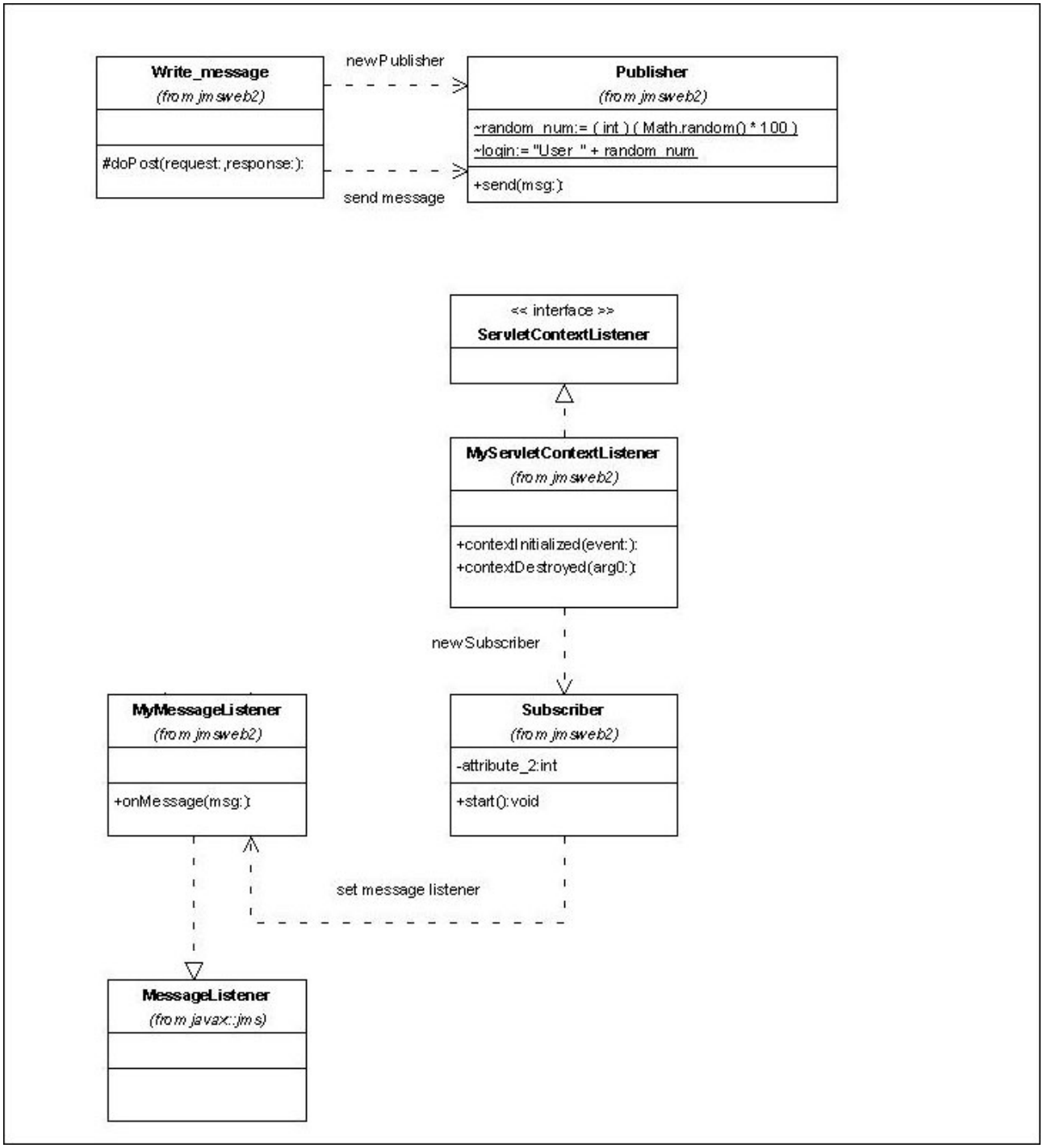
- DAO: il data access object da noi creato presenta 4 metodi che ci permettono di fare le principali operazioni sul nostro database.
    - store_msg: è il metodo che viene richiamato dal MyMessageListener quando la singola applicazione web riceve un messaggio, per memorizzarlo sul db. Al metodo viene passato il messaggio da memorizzare.
    - list_messages: viene richiamato dalla servlet ListMessages e serve per ottenere un vettore dei messaggi memorizzati nel nostro database. I messaggi sono rappresentati sotto forma di oggetti MessageData.
    - get_msg: viene richiamato dalla omonima servlet per ottenere un singolo messaggio presente sul db.
    - - erase_msg: viene richiamato dalla corrispondente servlet, passando l'id del messaggio che va cancellato sul db.

  La connessione viene stabilita, viene creato uno statement su cui è eseguita la query e una volta ottenuti i dati in risposta dal database o comunque terminate le operazioni da eseguire, la connessione viene chiusa.

Ogni metodo del DAO fa uso dei relativi driver per il DBMS MySQL usato nel nostro caso e di una connessione e uno statement che vengono ricreati ogni volta che viene usato uno dei metodi.

- MESSAGE DATA: questa classe rappresenta la struttura dati relativa a un messaggio. Questa struttura dati viene usata per memorizzare i messaggi in un vettore restituito dal metodo list_messages. La classe presenta solo l'id come attributo in quanto attualmente la nostra applicazione web deve mostrare ed usare solo esso sia per mostrare la lista dei messaggi tramite l'interfaccia web sia per leggere / cancellare il singolo messaggio.
- MY_MESSAGE_LISTENER: il my message listener è il nostro listener in corrispondenza dei messaggi che arrivano dal server Joram. Esso viene attivato dal Subscriber con l'avvio dell'applicazione web grazie al ServletContextListener. Il message listener ha il compito di memorizzare ogni messaggio che arriva tramite il metodo store_msg richiamato sul DAO.

4.4 PUBLISHER-SUBSCRIBER CLASS DIAGRAM



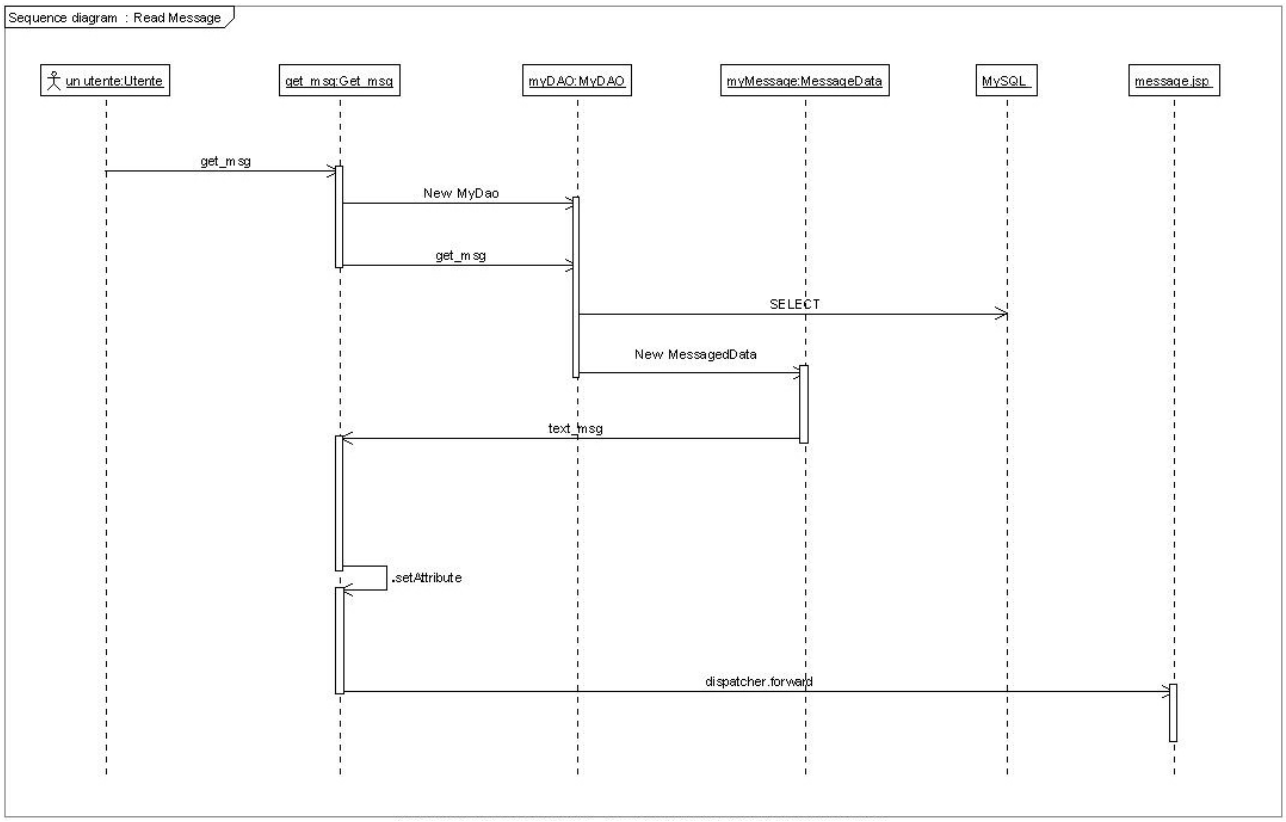
- MYSERVLET_CONTEXT_LISTENER: che implementa l'interfaccia ServletContextListener. Esso viene avviato con l'avvio della nostra applicazione web e grazie al metodo context initialized crea il Subscriber per sottoscriversi al Topic sul server Joram e crea il MessageListener.
- PUBLISHER: Viene creato dalla servlet write_message che richiama su esso il metodo send per spedire il messaggio. Si connette al Joram effettuando il lookup per la TopicConnectionFactory e per il topic sul quale spedirà il messaggio ricevuto tramite la servlet dal form html.
  Genera un login composto da un numero casuale. Questo login viene settato come proprietà di ogni messaggio che viene mandato al server Joram.

In questo modo ogni applicazione web ha un suo identificativo che si ipotizza, essendo casuale, unico in linea di massima.

- SUBSCRIBER: si connette al Joram effettuando il lookup e sottoscrivendosi al topic relativo, in modo durevole. Una volta effettuate queste operazioni costruisce un message listener associandolo al topic al quale si è sottoscritto, in modo che ogni volta che arrivi un messaggio esso venga memorizzato sul database. Quando un messaggio viene ricevuto, lo si confronta la sua proprietà relativa al login con quello generato nel proprio publisher. Questo perché ogni messaggio spedito a Joram viene inoltrato a tutte le applicazioni web compresa quella mittente. Di conseguenza si usa questo meccanismo per filtrare i messaggi in modo che il messaggio spedito dal mittente arrivi alle altre applicazioni web ma non a colui che lo ha spedito.

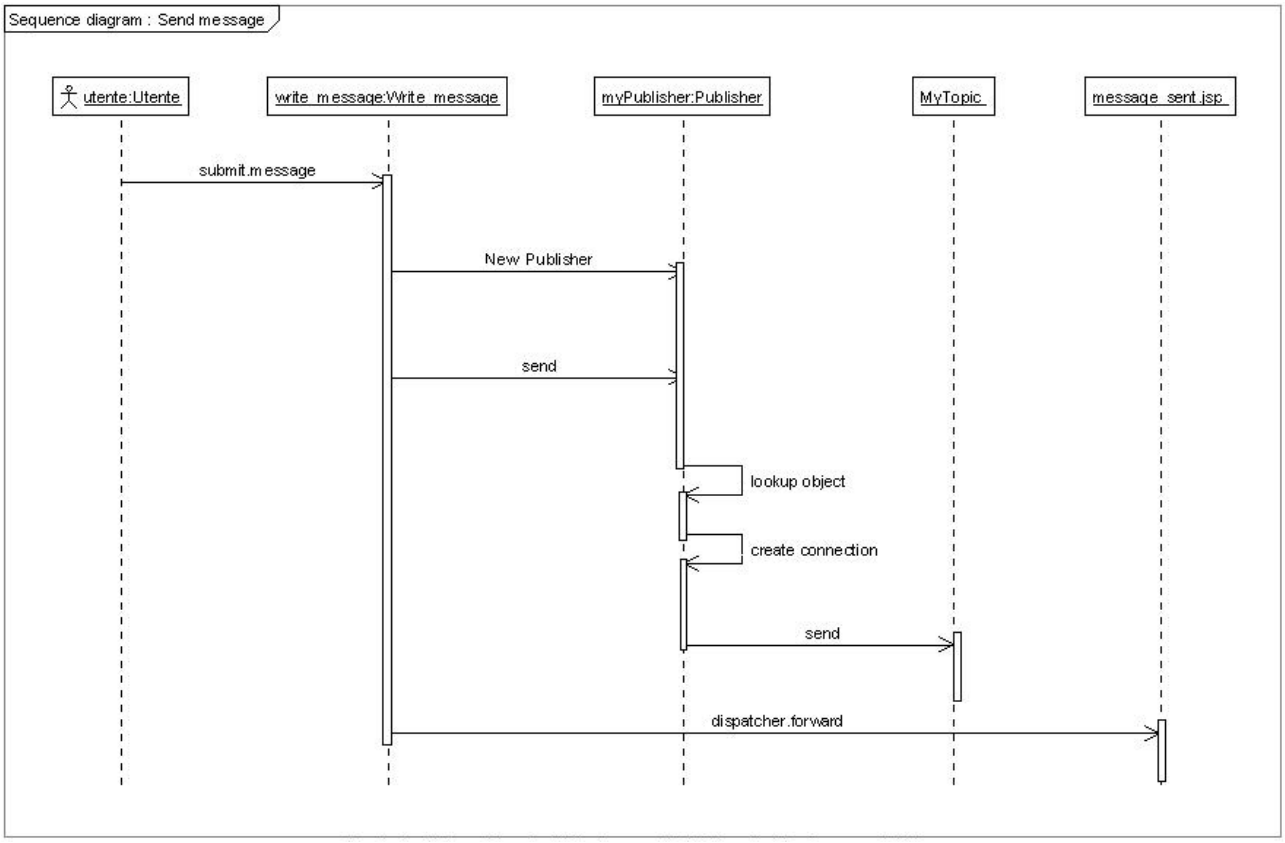
5 UML Sequence diagram

5.1 READ MESSAGE



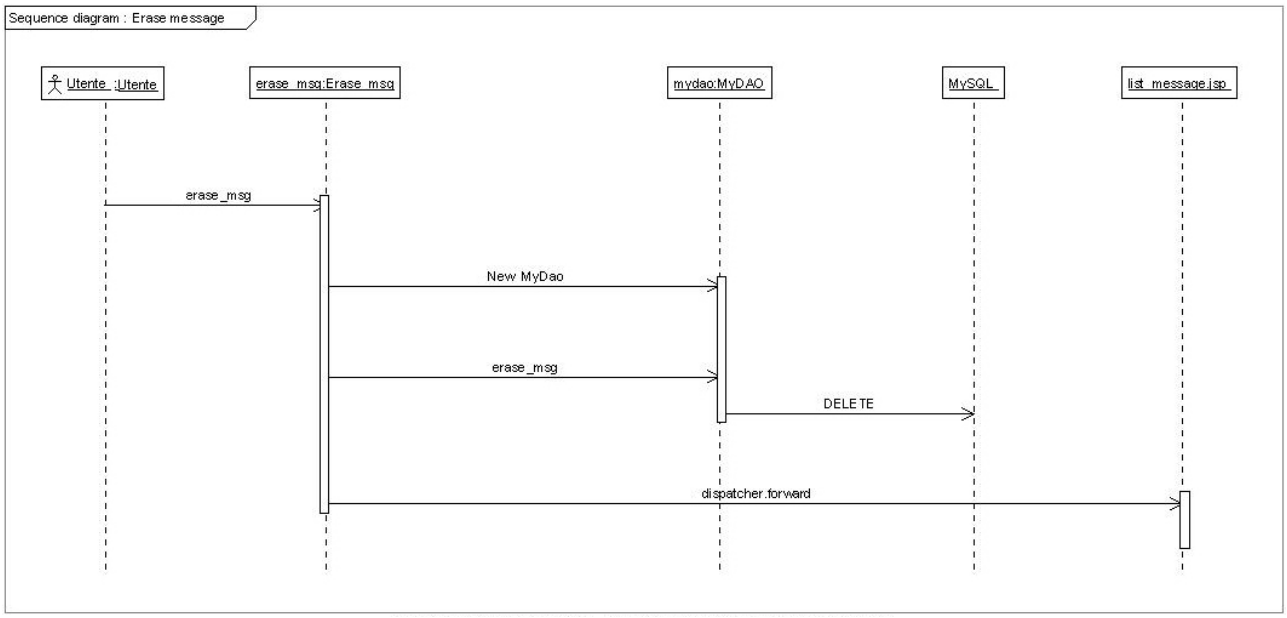
Lo Schema sopra riportato è il sequence diagram relativo alla lettura di un messaggio da parte di un utente.
L'attore clicca sulla relativa icona per leggere il messaggio che trova su ogni riga della lista dei messaggi.
Tramite questa azione passa l'id del messaggio alla servlet get_msg che crea un DAO sul quale applica il metodo get_msg passando ancora l'id e reperisce quindiil relativo messaggio sottoforma di MessageData dal DBMS MySQL.
Il messaggio vero è proprio è rappresentato da una stringa nell'oggetto restituito dal DAO.
Il testo ritornato viene settato come attributo del HttpServletRequest.
La servlet riporta il controllo alla pagina message.jsp che prende dal Request il testo del messaggio e lo mostra per farlo leggere all'attore.

5.2 SEND MESSAGE



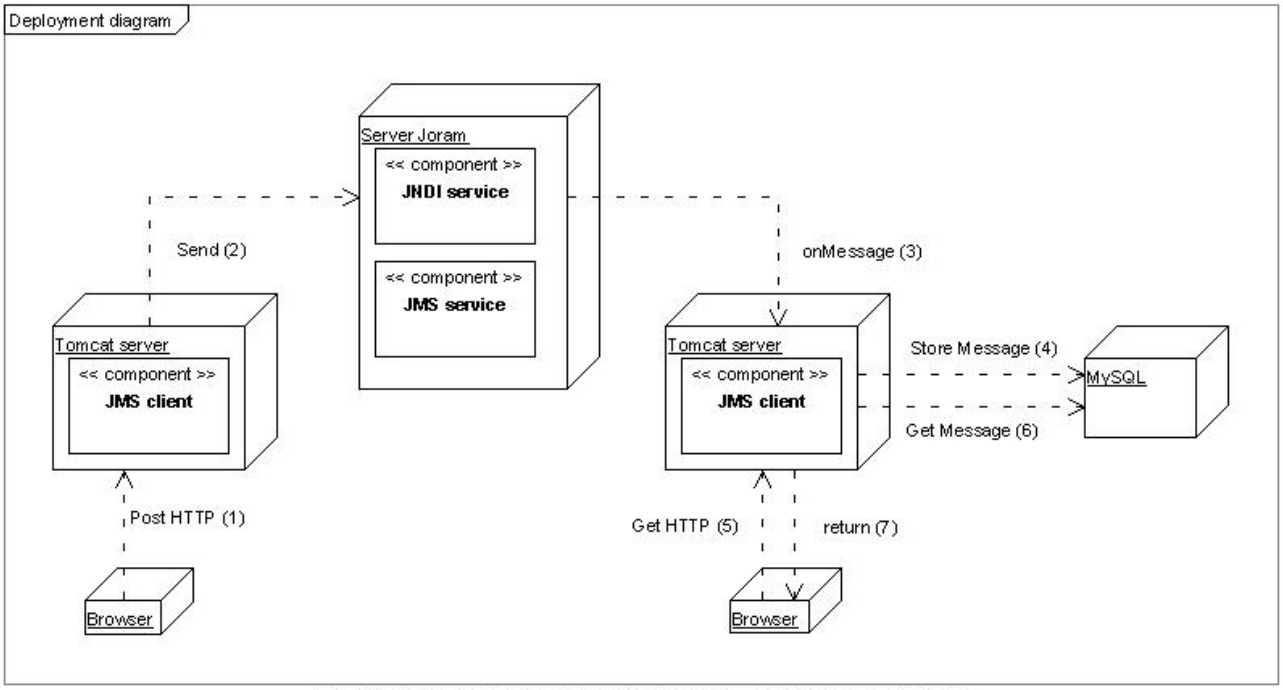
La figura sopra riportata rappresenta il diagramma di sequenza relativo al caso d'uso in cui l'utente spedisce un messaggio.
L'utente accedendo all'interfaccia web clicca sul relativo link e accede alla pagina che presente il form. Attraverso il modulo costituito dal form html l'utente scrive il messaggio e cliccando sul pulsante esso richiama una servlet che spedisce il messaggio tramite il publisher. Il publisher effettua il lookup e crea una connessione per pubblicare il messaggio relativamente al topic specificato. A questo punto la servlet write message restituisce il controllo a una specifica pagina che segnala all'utente la corretta riuscita dell'operazione tramite una notifica relativa al messaggio spedito.

5.3 ERASE MESSAGE



Lo schema sopra riportato è il sequence diagram relativo al caso d'uso nel quale l'utente cancella un messaggio.
L'attore richiede di cancellare un messaggio tramite il click sull'apposita icona, presente nella lista dei messaggi accanto ad ogni specifico messaggio.
Tramite il click si richiama la relativa servlet erase_msg alla quale viene passato l'id del messaggio da cancellare. Essa crea il DAO sul quale richiama il metodo erase_msg passando l'id. A questo punto il metodo del DAO effettua l'operazione di delete del messaggio sul DB, restituisce il controllo alla servlet che a questo punto mostra nuovamente la lista dei messaggi aggiornata, in quanto il messaggio precedente non è più presente.

6 Viste di Deploy



6.1 DEPLOY DIAGRAM

Lo schema sopra riportato è il deployment diagram relativo alla nostra enterprise application e mostra come ogni applicazione web sia presente su un server tramite il webserver apache tomcat che a sua volta si interfaccia con uno suo DBMS MySQL.
Si accede all'applicazione tramite il proprio browser ed essa tramite il web server comunica con il Server Joram che a sua volta comprende anche un server JNDI.

6.2 INSTALLAZIONE

Per installare e usare l'applicazione è necessario installare da prima il server Joram su una macchina che avrà il ruolo di server, dopo di che ogni client dovrà avere una sua web application presente sul proprio web server TomCat. Si deve quindi installare su ogni macchina client il TomCat e il DBMS MySQL oltre a un browser da usare per richiamare l'interfaccia web dell'applicazione.
Tramite un file war si può effettuare il deploy dell'applicazione web sul proprio application server, grazie ad esempio al pannello di amministrazione di TomCat.
Per la configurazione abbiamo scritto una comoda classe ServerAdmin.java che se fatta girare imposta i parametri sul server Joram.
Tramite questa procedura viene configurato anche il server JNDI.
Viene impostato l'IP del server e le relative porte.
Inoltre vengono create anche la TopicConnectionFactory e un Topic che verra usato dai subscriber.
Sul Topic vengono impostate le proprietà di free writing e free reading.

6.3 AVVIO

Una volta avviato il server bisogna configurarlo. Questo può essere fatto sia tramite una interfaccia grafica predisposta da Joram, sia tramite una classe di configurazione ServerAdmin.
A questo punto ogni client può avviare la propria applicazione che nel momento dell'avvio si collega al server Joram per sottoscriversi tramite il Subscriber.

7 Conclusioni

Questa enterprise application dimostra in modo semplice e trasversale ciò che si può fare tramite l'implementazione di poche classi Java usate tramite una interfaccia web per scambiare messaggi fra entità che vedono così suddividere la quantità di dati a loro destinata tramite la gestione dell'informazione scambiata che viene così resa permanente in modo individuale ottimizzando il processo di comunicazione.

Sostanzialmente il caso visto rappresenta un semplice modo per far comunicare più stazioni fra loro tramite uno scambio di messaggi che verranno memorizzati solo dal ricevente.
Tutto questo è reso possibile tramite gli strumenti visti: Java, Servlet, JSP, Joram, MySQL uniti nella nostra applicazione, che può essere estesa per un uso più professionale, efficace ed efficiente in un contesto di lavoro reale.

ESTENSIONE DEL PROGETTO
Il progetto può essere esteso in molti modi a cominciare da l'implementazione di un database centrale per gestire l'accesso e l'identificazione di ogni singola applicazione e dei suoi utenti.
Di conseguenza si dovrebbe modificare l'attuale applicazione web in modo da realizzare la relativa interfaccia per registrare nel sistema una nuova applicazione e gli utenti che la usano.
A questo punto ogni singolo utente potrebbe riconoscere nella lettura dei messaggi il mittente associandovi la relativa applicazione web e l'utente che ha inviato il messaggio.
Una applicazione pratica potrebbe essere quella di un gruppo di aziende legate fra loro, le quali tramite un sistema del genere hanno una piattaforma per la loro comunicazione interna.
Il sistema permetterebbe così di scambiare grosse quantità di messaggi facendo in modo che ogni singolo utente possa scegliere eventualmente quali aziende del gruppo o quali utente devono ricevere il messaggio.
Si può inoltre estendere la struttura del messaggio scambiato aggiungendovi altri campi come la data e l'ora di arrivo.