Specifica tecnica

Informazioni sul documento	
Nome file:	specifica_tecnica.2.0.pdf
Versione:	2.0
Data creazione:	2013-01-23
Data ultima modifica:	2013-02-20
Stato:	Approvato
Uso:	Esterno
Lista di distribuzione:	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Dott. Gregorio Piccoli
	Team SoftwareSynthesis
Redattori:	Andrea Rizzi
	Elena Zecchinato
	Marco Schivo
	Stefano Farronato
Approvato da:	Diego Beraldin
Verificatori:	Andrea Meneghinello

Storia delle modifiche

Versione	Descrizione intervento	Membro	Ruolo	Data
2.0	Approvazione del documento	Diego Beraldin	Responsabile	2013-02-20
1.10	Correzione errori segnalate dal verificatore	Andrea Rizzi	Progettista	2013-02-18
1.9	Verifica del documento	Andrea Meneghinello	Verificatore	2013-02-19
1.8	Aggiornamento diagrammi	Andrea Rizzi	Progettista	2013-02-18
1.7	Aggiornamento di alcuni presenter	Andrea Rizzi	Progettista	2013-02-18
1.6	Aggiornamento alcune classi server	Andrea Rizzi	Progettista	2013-02-15
1.5	Aggiornamento logica di rete	Stefano Farronato	Progettista	2013-02-12
1.4	Aggiornamento vantaggi del framework *Hibernate*	Marco Schivo	Progettista	2013-02-12
1.3	Correzione del modello dei dati	Marco Schivo	Progettista	2013-02-11
1.2	Aggiunta descrizione dei diagrammi di attività e correzione degli stessi	Stefano Farronato	Progettista	2012-02-09
1.1	Modifica architettura del database	Elena Zecchinato	Progettista	2013-02-08
1.0	Approvazione documento	Riccardo Tresoldi	Responsabile	2013-01-30
0.19	Correzione errori ortografici e di forma presenti nel documento in base alle segnalazioni del verificatore	Marco Schivo	Progettista	2013-01-29
0.18	Correzione diagrammi presenti nel documento in base alle segnalazioni del verificatore	Elena Zecchinato	Progettista	2013-01-29
0.17	Verifica lessico ortografica del documento	Andrea Meneghinello	Verificatore	2013-01-28
0.16	Verifica correttezza e corrispondenza dei diagrammi presenti nel documento	Stefano Farronato	Verificatore	2013-01-27
0.15	Inserimento tabelle di tracciamento prodotte nel capitolo 12	Diego Beraldin	Progettista	2013-01-28
0.14	Inserimento diagrammi delle attività e dei package prodotti nel capitolo 11	Diego Beraldin	Progettista	2013-01-28
0.13	Inserimento diagrammi delle classi nel capitolo 6, 7, 8, 9	Marco Schivo	Progettista	2013-01-28
0.12	Inserimento diagrammi relativi ai design pattern evidenziati	Marco Schivo	Progettista	2013-01-28
0.11	Inizio stesura capitolo relativo alla descrizione delle classi	Diego Beraldin	Progettista	2013-01-27
0.10	Stesura della sezione relativa all'architettura `mytalk.clientview` stilando i componenti evidenziati	Elena Zecchinato	Progettista	2013-01-27

0.9	Stesura della sezione relativa all'architettura `mytalk.clientpresenter` stilando i componenti evidenziati	Marco Schivo	Progettista	2013-01-27
0.8	Stesura della sezione relativa all'architettura `mytalk.server` stilando i componenti evidenziati	Marco Schivo	Progettista	2013-01-26
0.7	Stesura della sezione relativa alla progettazione logica	Diego Beraldin	Progettista	2013-01-26
0.6	Completata la sezione relativa alla progettazione concettuale	Elena Zecchinato	Progettista	2013-01-25
0.5	Inizio stesura della sezione relativa alla progettazione concettuale con classi evidenziate in fase di progettazione	Elena Zecchinato	Progettista	2013-01-24
0.4	Descrizione dei design pattern evidenziati nella fase di progettazione.	Diego Beraldin	Progettista	2013-01-24
0.3	Aggiunto capitolo relativo agli strumenti utilizzati.	Marco Schivo	Progettista	2013-01-23
0.2	Stesura dell'introduzione ai design pattern. Stesura dell'introduzione ai tracciamenti.	Elena Zecchinato	Progettista	2013-01-23
0.1	Creazione del documento e stesura della sezione "Introduzione".	Diego Beraldin	Progettista	2013-01-23

Indice

Elenco delle figure

Sommario

Il presente documento illustra l'architettura del sistema ad alto livello e comprende una suddivisione di quest'ultima in sotto-architetture logiche, l'enumerazione e la descrizione dei componenti funzionali che le costituiscono nonché la decomposizione dei componenti nei package e, in ultima istanza, nelle classi. Vengono presentati inoltre gli elementi di riuso architetturale (*pattern*) utilizzati, motivandone la scelta. Infine, il documento è corredato da sei tabelle riepilogative inerenti al tracciamento componenti-requisiti, componenti-*design pattern* e componenti-classi.

1 Introduzione

1.1 Scopo del prodotto

Con il progetto "MyTalk" si intende un sistema software di comunicazione tra utenti mediante browser senza la necessità di installazione di plugin e/o software esterni. L'utilizzatore avrà la possibilità di interagire con un altro utente tramite una comunicazione audio - audio/video - testuale e, inoltre, ottenere delle statistiche sull'attività in tempo reale.

1.2 Scopo del documento

Il presente documento è stato redatto al fine di produrre le specifiche sulla progettazione ad alto livello, del prodotto MyTalk. A tal fine il documento presenterà:

- una descrizione degli strumenti e dei *framework* su cui si basa l'architettura;
- un elenco con le specifiche dei *design pattern* utilizzati;
- l'architettura di alto livello del sistema;
- una descrizione dettagliata dei componenti rilevati in fase di progettazione indicando relativamente a ciascuno di essi il tipo, la funzione e l'obiettivo;
- i diagrammi UML per definire i flussi principali di controllo dell'applicativo;
- il tracciamento di requisiti e componenti;
- il tracciamento di componenti e *design pattern*;
- il tracciamento di classi e componenti.

1.3 Glossario

Al fine di evitare incomprensioni dovute all'uso di termini tecnici nei documenti, viene redatto e allegato il documento *glossario.3.0.pdf* dove vengono definiti e descritti tutti i termini marcati con una sottolineatura.

2 Riferimenti

2.1 Normativi

piano_di_qualifica.3.0.pdf allegato.

norme_di_progetto.3.0.pdf allegato.

analisi_dei_requisiti.3.0.pdf allegato

2.2 Informativi

Capitolato d'appalto: MyTalk, v1.0, redatto e rilasciato dal proponente Zucchetti S.r.l. reperibile all'indirizzo `http://www.math.unipd.it/~tullio/IS-1/2012/Progetto/C1.pdf`;

testo di consultazione: *Software Engineering (8th edition) Ian Sommerville, Pearson Education / Addison Wesley*;

manuale all'utilizzo dei design pattens: *Design Patterns, Elementi per il riuso di software a oggetti – (1/Ed. italiana) Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, Pearson Education*;

manuale di basi di dati: *Sistemi di basi di dati-fondamenti* – (6° edizione) Ramez Elmasri / Shamkant B. Navathe

glossario.3.0.pdf allegato.

3 Strumenti utilizzati

3.1 Java

L'utilizzo del linguaggio Java è richiesto dal proponente esclusivamente per la realizzazione della componente server.

Vantaggi

- è un linguaggio predisposto nativamente alla gestione parallela di *thread* e questo applicato ad un server dà la possibilità di gestire parallelamente richieste da parte di più utenti allo stesso tempo;
- essendo un linguaggio orientato agli oggetti e fortemente tipizzato si presta all'applicazione di *design pattern* e alla costruzione di un'architettura robusta, fortemente modulare e al contempo flessibile, in accordo con i principi del paradigma di programmazione OO;
- permette la generazione automatica della documentazione con l'ausilio di JavaDoc;
- garantisce la portabilità del codice (a livello di bytecode), l'indipendenza dalla piattaforma fisica di esecuzione grazie alla JVM e l'integrazione nell'ambiente di esecuzione del proponente (TomCat).

3.2 Web server Apache

La struttura dell'applicativo si basa su pagine web scritte in HTML5. Ciò richiede la presenza di un server web dove collocare le pagine richiamabili dai browser degli utenti.

Vantaggi

- diversi membri del team hanno già lavorato su tale struttura, il costo in termini di tempo per istruire il personale è dunque ridotto al minimo;
- Apache offre funzioni di controllo per la sicurezza come quelli che compie il *proxy*;

3.3 RDBMS MySQL

Ogni utente avrà a disposizione diversi dati consultabili e modificabili in diverse aree dell'applicativo. Per esempio, un utente sarà interessato a gestire i propri dati personali.

Ciò richiede che l'applicativo sia dotato di un sistema di memorizzazione permanente dei dati. A tale scopo si è scelto di usare un RDBMS (*relational database management system*) e nello specifico il team intende appoggiarsi ad un server MySQL.

Vantaggi

- come per Apache, anche per questo sistema si stima che costi per istruire i personali saranno minimi, poiché i membri del team hanno già avuto modo di utilizzare tale strumento;
- uno dei principali strumenti d'amministrazione di un database MySQL è phpMyAdmin. Il team conta di utilizzarlo in virtù dell'intuitività con cui è costruita l'interfaccia grafica;
- tale DBMS supporta le transazioni, essenziali nel nostro progetto per la manipolazione e la lettura dei dati memorizzati.

3.4 Servlet container TomCat

L'applicativo dovrà disporre di un lato server ed un lato client. Se da una parte il lato client è rappresentato dall'ambiente di esecuzione del browser che ha il compito di visualizzare una pagina web fornita tramite un server Apache ed eseguire gli *script* ad essa associati, dall'altra

l'applicativo server sarà strutturato tramite delle *servlet* Java che, per conformità con i software usati, saranno caricate nel *servlet container* di TomCat che si appoggia sul già citato Apache.

Vantaggi

- Alcuni membri del gruppo hanno già lavorato in passato con Tomcat; la loro esperienza potrà tornare utile velocizzando la fase d'apprendimento, da parte dei membri che ancora non lo conoscono;
- contiene le librerie WebSocket.

3.5 Hibernate

Hibernate è un *framework* Java utilizzato per facilitare l'utilizzo di un database da parte del server realizzando la mappatura fra oggetti intesi in senso OOP ed ennuple del modello relazionale (*object-relational mapping*).

Vantaggi

- Hibernate permette di utilizzare le tabelle di un database relazionale come se fossero degli oggetti mappando il database su di opportune classi strutturate ad-hoc svincolando la gestione della persistenza dei dati dalla logica di business;
- con questo *framework* Java riesce a lavorare su un database rendendo trasparenti al programmatore le vere e proprie *query* e mostrando esclusivamente classi e metodi; Lo sviluppatore è inoltre esonerato dalla gestione dei risultati di chiamate SQL e la loro eventuale conversione in oggetti.
- essendo rilasciato sotto licenza LGPL può essere utilizzato senza restrizioni (*copyleft*) e vincoli di licenza delle opere derivate.
- l'applicazione rimane portabile in tutti i sistemi di gestione supportati, con un basso *overhead*.

3.6 JAAS

JAAS (*Java Authentication and Authorization Service*) è un *framework* Java utilizzato per la gestione della sicurezza.

Vantaggi

- Permette di gestire la gestire la logica di *login* indipendentemente dalla struttura dell'applicativo sottostante;
- la manutenzione incorpora lo stesso vantaggio presentato al punto precedente: indipendenza dall'applicativo che ne fa uso;
- facilità nel gestire autenticazione e autorizzazione.

3.7 WebRTC

WebRTC è il *framework* JavaScript richiesto da capitolato per gestire le comunicazioni VOIP.

Vantaggi

- Facilita l'iter di connessione e gestione della comunicazione, evitando l'utilizzo diretto delle WebSocket;
- riduce il *total cost of ownership* nella creazione di applicativi VOIP (poiché non richiede l'installazione di componenti aggiuntivi e si basa esclusivamente sul browser da cui è lanciato l'applicativo).

4 Introduzione all'architettura di sistema

Per introdurre l'architettura proposta è essenziale mettere in evidenza le seguenti considerazioni:

- il sistema poggia su un database nel quale sono contenuti i dati d'interesse per gli utenti (dati anagrafici, lista dei messaggi audio, lista dei messaggi audio e video, la rubrica dei contatti e lo storico delle chiamate);
- il sistema proposto dal team è dotato di una parte server e una parte client;
- dopo un'analisi preliminare il team ha stabilito che la progettazione del server non deve essere vincolata da quella del client in modo tale da evitare che il progetto dell'applicativo lato server abbia la cognizione di come funziona il client. Ciò permetterà un futuro riutilizzo del codice (e.g. se si desiderasse creare un nuovo applicativo di tipo VoIP si potrà riutilizzare il server già creato);
- per quanto riguarda il lato client, al fine di garantire un alto livello di riutilizzo del codice e la possibilità di eseguire manutenzioni nel minor tempo possibile, si vuole che la logica d'implementazione del client sia svincolata dalla rappresentazione grafica del medesimo.

Tali considerazioni di base, hanno portato il team a suddividere l'architettura in tre sotto-architetture, intese anche come package, più una quarta architettura inerente la struttura del database:

- il database;
- il server (`org.softwaresynthesis.mytalk.server`);
- il presenter del client (`org.softwaresynthesis.mytalk.clientpresenter`);
- la vista del client (`org.softwaresynthesis.mytalk.clientview`).

Le specifiche di ogni sotto-architettura saranno definite, rispettivamente nelle sezioni 5 a pagina 8, 6 a pagina 13, 7 a pagina 38 e 8 a pagina 49.

Inoltre si fa presente che l'architettura generale, intesa come agglomerato delle tre sotto-architetture precedentemente elencate, fa uso del pattern MVP. Con tale ottica la sotto-architettura `server` ricopre il ruolo di *model*, `clientpresenter` costituisce invece il *presenter*, mentre `clientview` è la vista definita per questo progetto. Tra le considerazioni più interessanti che hanno portato alla scelta di questo *pattern*, va messa in evidenza la seguente:

assegnando a ogni sotto-architettura un ruolo specifico, si garantisce un alto livello di riutilizzo del codice (e.g. la vista comunica con il *presenter* poiché non conosce la logica di *business* del sistema).

Di conseguenza, in un futuro si potrebbe riprendere la vista oggi definita e riutilizzarla in un altro progetto, andando solo a ridefinire, se necessario, un *presenter* che riproponga una nuovo adattamento della parte logica.

Nelle sezioni successive, per ogni sotto-architettura sarà fornito un elenco dettagliato dei componenti che lo interessano. Si sottolinea che per componenti non sono da intendersi i sotto-package, bensì gli agglomerati di classi (potenzialmente appartenenti a package diversi) che concorrono ad un fine comune: la definizione delle funzionalità del componente trattato.

4.1 Logica di rete

4.1.1 Descrizione della logica di connessione

La rete che viene a crearsi sotto l'architettura di MyTalk è basata su WebRTC e WebSocket. Ogni client connesso al sistema ha una canale WebSocket con il server e per ogni chiamata che viene a crearsi tra due client si instaura un canale WebRTC che mette in comunicazioni i seuddetti client. Questo canale è generato dall'oggetto RTCPeerConnection che è alla base dell'API di WebRTC.

Una comunicazione nasce tra due utenti, colui che effettua la chiamata e colui che la riceve.

1. I client che possono effettuare o ricevere un chiamata sono tutti quelli che sono connessi al server con una WebSocket e sono indicizzati all'interno di un array attraverso un ID univoco;
2. Al momento di effettuare una chiamata, il chiamante invia una richiesta al server attraverso il canale WebSocket già aperto, comunicando l'ID del chiamato e la propria *description* (ovvero un elenco di caratteristiche e parametri necessari alla comunicazione WebRTC, tra cui il socket pubblico);
3. Il server usa il canale WebSocket già aperto con il chiamato per comunicargli l'intenzione del chiamante di instaurare una comunicazione e recapitare la *description* di quest'ultimo;
4. Il chiamato, dopo aver accettato la chiamata dando il consenso all'utilizzo della webcam, riceve la *description* del chiamante, la imposta nel proprio oggetto webRTCPeerConnection, sucessivamente invia al server la propria *description*;
5. Il server comunica la *description* del chiamato al chiamante e termina il proprio ruolo all'interno della comunicazione;
6. Il chiamante riceve la *description* del chiamato e la setta nel proprio oggetto webRTCPeerConnection;
7. Giunti a questo punto gli oggetti webRTCPeerConnection di entrambi i client hanno tutte le *description* che servono per avviare la comunicazione *peer-to-peer*;
8. La comunicazione rimarrà attiva fino a che uno dei due client non deciderà volutamente di interromperla.

5 Architettura del database

Come già citato nella parte introduttiva all'architettura, il server si appoggia ad un database dove sono registrati i dati del sistema. La progettazione di tale database passa per due fasi (come appreso dal manuale "Sistemi di basi di dati"): la progettazione concettuale e la progettazione logica.

5.1 Progettazione concettuale

Lo scopo della progettazione concettuale è definire una struttura "concettuale" della base di dati.

Tale struttura non rappresenta quella finale pronta per la creazione sul DBMS, ma bensì un modello in grado di rappresentare il problema, svincolato da come esso debba essere logicamente rappresentato.

5.1.1 Lista delle classi

UserData: è l'entità le cui istanze rappresentano gli utenti registrati nel sistema. UserData è caratterizzata dai seguenti attributi:

Nome attributo	Tipo	Opzionale	Vincoli
E_Mail	Varchar(100)	NO	Unique
Password	Varchar(100)	NO	
Question	Varchar(100)	NO	
Answer	Varchar(100)	NO	
Name	Varchar(100)	SÌ	
Surname	Varchar(100)	SÌ	
Picture	Varchar(150)	NO	

AddressBookEntries: è l'entità le cui istanze rappresentano i contatti appartenenti alla rubrica di un utente. AddressBookEntries è caratterizzata dai seguenti attributi:

Nome attributo	Tipo	Opzionale	Vincoli
Blocked	Boolean	NO	default(false)

Calls: è l'entità le cui istanze rappresentano le chiamate effettuate attraverso il sistema software MyTalk. Calls è caratterizzata dai seguenti attributi:

Nome attributo	Tipo	Opzionale	Vincoli
Start_date	Timestamp	NO	default(current_timestamp)
End_date	Timestamp	SÌ	

Groups: è l'entità le cui istanze rappresentano un gruppo della rubrica di un utente. Tale entità è costituita dagli attributi:

Nome attributo	Tipo	Opzionale	Vincoli
Name	Varchar(100)	NO	

Messages: è l'entità che rappresenta le informazioni basilari di un messaggio della segreteria dell'utente. Gli attributi che la compongono sono:

Nome attributo	Tipo	Opzionale	Vincoli
New	Boolean	NO	
Start_date	Timestamp	NO	default(current_timestamp)

AudioMessages: è un'entità che specializza Messages e rappresenta i messaggi audio lasciati nella segreteria di un utente. L'entità non è caratterizzata da alcun attributo proprio.

AudioVideoMessages: è un'entità che specializza Messages e rappresenta i messaggi audio/video lasciati nella segreteria di un utente. L'entità non è caratterizzata da alcun attributo proprio.

5.1.2 Gerarchia tra classi

Messages (AudioMessages e AudioVideoMessages): i messaggi si suddividono logicamente in due categorie, i messaggi dotati solamente di traccia audio e quelli aventi anche una traccia video. Gli attributi contenuti in queste entità sono gli stessi.

Infatti la necessità di mostrare la separazione deriva dalla volontà del team di evidenziare la distinzione tra i due oggetti, motivata dalla possibilità di gestire le istanze delle due entità in modo diverso.

5.1.3 Associazione tra classi

UserData - Calls (molti a molti): tale associazione rappresenta il legame che intercorre tra gli utenti e le chiamate. Tale associazione è del tipo molti a molti, con totalità parziale verso Calls e totalità totale verso UserData. L'associazione prevede l'attributo 'Caller' che identifica chi è stato il primo utente ad avviare la chiamata.

UserData - AddressBookEntries (uno a molti): tale associazione rappresenta il legame che intercorre tra un utente, il possessore della rubrica, e l'insieme dei contatti presenti in essa. Tale associazione è di tipo uno a molti, con totalità totale verso Userdata e parziale verso AddressBookEntries.

AddressBookEntries - UserData (uno a uno): Tale associazione rappresenta il proprietario, univoco, di ogni contatto della rubrica. Tale associazione è del tipo uno a uno con totalità parziale da UserData verso AddressBookEntries mentre ha totalità totale da AddressBookEntries verso UserData.

AddressBookEntries - Groups (molti a uno): tale associazione rappresenta la possibilità di suddividere la propria rubrica in gruppi di contatti. Tale associazione è del tipo molti a uno, con totalità totale da Groups ad AddressBookEntries mentre ha una totalità parziale da AddressBookEntries a Groups.

UserData - Messages (uno a molti): tali associazioni rappresentano il legame che intercorre tra gli utenti ed i messaggi registrati nella propria segreteria. Le associazioni sono del tipo uno a molti verso Messages, con totalità parziale verso Messages e totalità totale verso UserData. L'idea alla base è che un utente può avere nella propria segreteria uno o più messaggi, così come può non averne nessuno, mentre ogni messaggio della segreteria è associato a un mittente e a un destinatario.

5.2 Progettazione logica

La progettazione logica ha la finalità di costruire la struttura definitiva del database a partire da quanto stabilito in sede di progettazione concettuale.

Pertanto, si deciderà come trasformare le gerarchie e in seguito le associazioni, al fine di restituire una struttura chiara e pronta per la creazione sul DBMS. La lista delle classi proposta e lo schema associato, rappresentano tale struttura.

5.2.1 Rappresentazione delle gerarchie

Messages(AudioMessages e AudioVideoMessages): al fine di evidenziare la suddivisione logica tra le due tipologie di messaggi, è stato deciso di trasformare la specializzazione di Messages con una tabella unica contenente un discriminante. È rimasta esclusivamente l'entità Messages a cui è stato aggiunto l'attributo "Video" con la funzione di discriminare le due tipologie di messaggi.

5.2.2 Rappresentazione delle associazioni

UserData - Calls: tale associazione si è risolta attraverso l'introduzione dell'entità CallLists. Tale entità contiene l'attributo ID_user come chiave esterna verso l'entità UserData e l'attributo ID_call come chiave esterna verso l'entità Calls. È dotata inoltre dell'attributo "Caller" che identifica colui che ha per primo avviato la chiamata; per ogni gruppo di ennuple aventi lo stesso ID_call si avrà soltanto una ennupla contrassegnata con il campo Caller a vero.

UserData - AddressBookEntries: tali associazioni si sono risolte inserendo nell'entità AddressBookEntries due chiavi esterne verso UserData.

AddressBookEntries - Groups: tale associazione si è risolta inserendo una chiave esterna verso Groups all'interno dell'entità AddressBookEntries.

UserData - Messages: tale associazione si è risolta inserendo due chiavi esterne verso UserData all'interno dell'entità Messages.

5.2.3 Chiavi primarie sintetiche

I progettisti hanno optato per la definizione di chiavi primarie sintetiche, ovvero chiavi generate automaticamente all'inserimento di un nuovo *record* nelle tabelle corrispondenti alle entità.

Tale scelta è dettata dall'utilizzo del framework Hibernate che non permette la modifica del valore della chiavi primarie associate ad una istanza di un oggetto di *business*. Le chiavi saranno quindi composte da un campo a valore intero, da intendersi come un contatore che identifica univocamente i record.

La chiave sintetiche adottate sono del tipo `BIGINT` senza segno, il team è consapevole che questo impone un limite di $1,8 \times 10^{19}$ al numero di voci che possono essere memorizzate contemporaneamente in ogni entità.

Una soluzione da adottarsi in futuro per evitare la perdita di dati potrebbe consistere nello spostare da database a disco parte di queste informazioni. L'introduzione di questo tipo di chiave è dovuta anche ad una semplice gestione, tramite Hibernate, di tale oggetto.

Nel caso particolare dei messaggi, inoltre, il campo ID_message, oltre a rappresentare un identificatore univoco, coincide anche con il nome del file sul server in cui è memorizzato il messaggio stesso, all'interno di una *directory* predefinita e nota ai componenti del sistema.

5.2.4 Lista delle classi

UserData			
Nome attributo	Tipo	Opzionale	Vincoli
ID_user	Bigint unsigned	NO	PrimaryKey
E-Mail	Varchar(100)	NO	Unique
Password	Varchar(100)	NO	
Question	Varchar(100)	NO	
Answer	Varchar(100)	NO	
Name	Varchar(100)	SÌ	
Surname	Varchar(100)	SÌ	
Picture	Varchar(150)	NO	

Calls			
Nome attributo	Tipo	Opzionale	Vincoli
ID_call	Bigint unsigned	NO	PrimaryKey
Start_date	Datetime	NO	default(current_timestamp)
End_date	Datetime	SÌ	

CallLists			
Nome attributo	Tipo	Opzionale	Vincoli
ID_callList	Bigint unsigned	NO	PrimaryKey
ID_call	Bigint unsigned	NO	ForeignKey verso Calls
ID_user	Bigint unsigned	NO	ForeignKey verso UserData
Caller	Boolean	NO	default(false)

AddressBookEntries			
Nome attributo	Tipo	Opzionale	Vincoli
ID_addressBookEntry	Bigint unsigned	NO	PrimaryKey
ID_user	Bigint unsigned	NO	ForeignKey verso UserData
ID_group	Bigint unsigned	SI	ForeignKey verso Groups
Owner	Bigint unsigned	NO	
Blocked	Boolean	NO	default(false)

Groups			
Nome attributo	Tipo	Opzionale	Vincoli
ID_group	Bigint unsigned	NO	PrimaryKey
Name	Varchar(100)	NO	
Owner	Bigint unsigned	NO	ForeignKey verso UserData

Messages			
Nome attributo	Tipo	Opzionale	Vincoli
ID_message	Bigint unsigned	NO	PrimaryKey
Sender	Bigint unsigned	NO	ForeignKey verso UserData
Receiver	Bigint unsigned	NO	ForeignKey verso UserData
New	Boolean	NO	default(true)
Video	Boolean	NO	default(false)
Start_date	Datetime	NO	default(current_timestamp)

5.2.5 Diagramma delle classi

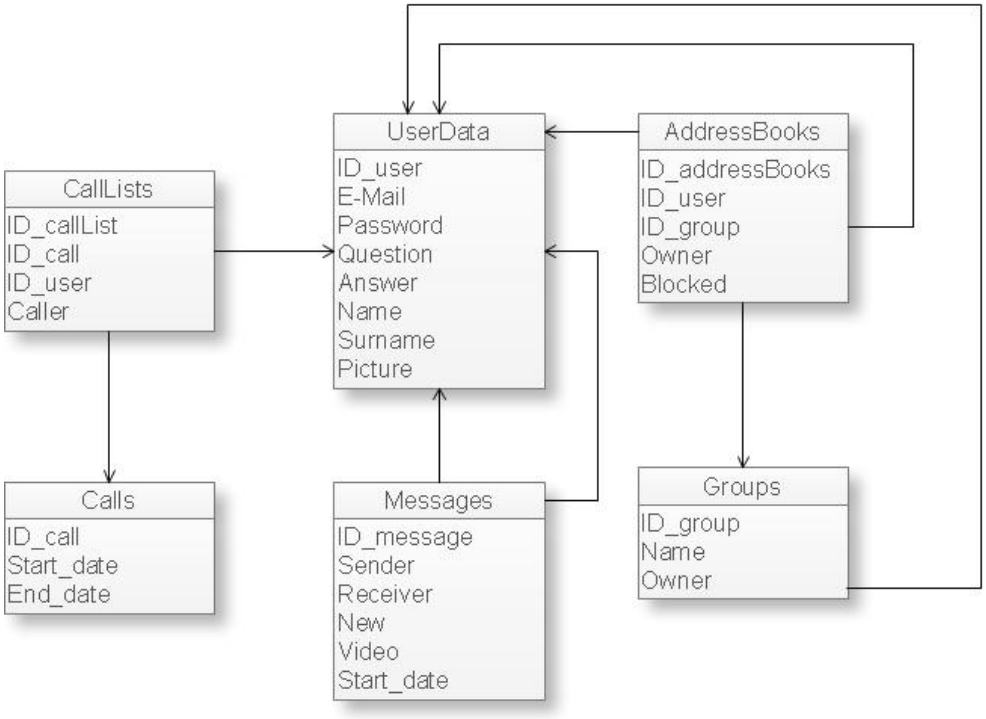

Figura 1: Diagramma delle classi - Schema logico database

6 Architettura mytalk.server

Tale sotto-architettura definisce le specifiche e le funzionalità dell'applicativo lato server. In esso saranno definiti i seguenti componenti:

- CS01 – Gestione database;
- CS02 – Gestione connessione;
- CS03 – Gestione rubrica;
- CS04 – Gestione autenticazione;
- CS05 – Gestione segreteria;
- CS06 – Gestione chiamate;
- CS07 – Façade del server.

I componenti sopracitati verranno definiti rispettivamente nelle sottosezioni 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6 e 6.1.7. Si sottolinea sin da ora che il server è l'unico in grado di comunicare con il database su cui si poggia l'applicativo.

Infine si fa notare che i nomi di tutte le classi riportate nella presente sezione sono implicitamente parte del package `org.softwaresynthesis.mytalk.server`, pertanto tale prefisso sarà omesso nella loro denominazione.

6.1 Componenti evidenziati

6.1.1 CS01 – Gestione database

Descrizione:

Tale componente si occupa di rappresentare la struttura del database relazionale su cui poggia l'applicativo, tramite esso il sistema potrà quindi effettuare operazione di lettura e scrittura di entità all'interno del database.

Nucleo logico di tale componente è la classe Singleton `HibernateUtil`, il cui compito è implementare una *factory* per le sessioni verso il database. Tali sessioni si basano sul file di configurazione di Hibernate, tra le cui informazioni compare anche la tipologia di DBMS utilizzata.

Le classi che fanno parte di questo componente sono quelle che costituiscono i *data access object* (DAO). Tali classi hanno il compito di inoltrare richieste per la modifica e la lettura dei dati (*Create*, *Read*, *Update* e *Delete*) nelle corrispondenti entità del database.

Tali classi sono:

- `dao.AddressBookEntryDAO`
- `dao.CallDAO`
- `dao.GroupDAO`
- `dao.MessageDAO`
- `dao.UserDataDAO`
- `dao.CallListDAO`

Per chiarezza citiamo in tale sezione anche le classi che mappano le entità rappresentate nel database. Tali sono dotate di variabili di istanza che corrispondono ai campi dei *record*. Le operazioni disponibili su questo genere di oggetti comprendono i metodi *get*/*set* associati ai campi delle tabelle del database e sono rese disponibili dalle interfacce (una per ogni entità) implementate dalle classi.[1]

- `abook.IGroup`

[1] Per chiarezza, si precisa che tali classi corrispondono a quelle che nella logica di programmazione in Hibernate prendono il nome di *Transfer Object* (TO).

- abook.Group
- abook.IAddressBookEntry
- abook.AddressBookEntry
- abook.IUserData
- abook.UserData
- call.ICall
- call.Call
- call.ICallList
- call.CallList
- message.IMessage
- message.Message

Precisiamo che le classi rappresentanti i *transfer object* si trovano a essere collocate in package e componenti diversi da server.dao. Si è ritenuto infatti sensato inserire tali classi in contesti maggiormente specializzati, ad esempio la classe corrispondente ai messaggi in segreteria verrà ad essere collocata in server.message, mentre quella corrispondente alle chiamate in server.call.

Diagramma delle classi:

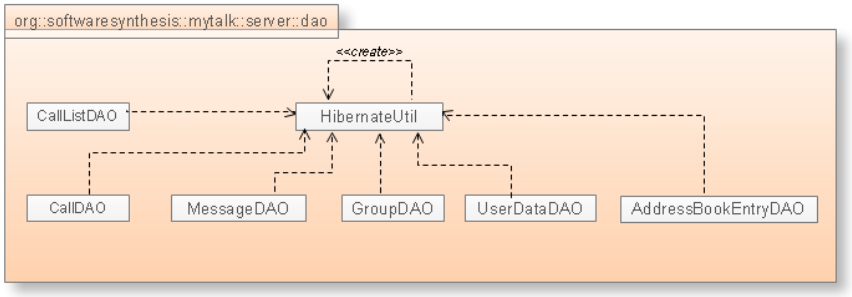

Figura 2: Diagramma delle classi - Gestione database

Classi utilizzate:

- `server.dao.CallDAO`
- `server.dao.CallListDAO`
- `server.dao.GroupDAO`
- `server.dao.MessageDAO`

- server.dao.UserDataDAO
- server.dao.AddressBookEntryDAO
- server.dao.HibernateUtil

6.1.2 CS02 – Gestione connessione

Descrizione:

Tale componente ingloba le classi destinate a stabilire le *routine* di connessione fra il server e i client del sistema, necessarie in un secondo momento a gestire le chiamate in ingresso e in uscita dai client.

La comunicazione bidirezionale fra il server e i client è ottenuta estendendo la classe fornita dalle librerie di *TomCat* `org.apache.catalina.websocket.MessageInbound` mediante `connection.PushInbound`. In particolare sarà realizzato l'*overriding* del metodo `onTextMessage(CharBuffer)` in modo da permettere al componente Gestione connessione del server di reagire correttamente ai messaggi provenienti dal client in forma testuale.

Un ruolo essenziale è inoltre svolto dalla *servlet* `connection.ChanelServlet`, realizzata estendendo mediante ereditarietà la classe fornita dalle librerie di *Apache TomCat* `org.apache.catalina.websocket.WebSocketServlet`. Tale *servlet* non fa parte di questa componente, viene qui citata unicamente per chiarire l'interazione tra il client e il server in merito alla comunicazione. La *servlet* risiede, come si vedrà in seguito, nella componente CS07 – Gestione Façade.

La caratteristica fondamentale della *servlet*, che rappresenta inoltre un punto di accesso centralizzato alle funzionalità del componente Gestione connessione, è quella di fare *overriding* del metodo `createWebSocketInbound(String, HttpServletRequest)` al fine di restituire un opportuno sottotipo di `org.apache.catalina.websocket.StreamInbound`. Il tipo dinamico dell'oggetto ritornato da `connection.ConnectionManager` corrisponde infatti proprio a `connection.PushInbound` in modo che lato client l'invocazione del costruttore di `WebSocket` (passando come parametro l'URL della *servlet*) crei una connessione del tipo voluto.

Ne consegue che, come sarà esposto con maggiori dettagli nella sezione 9.3 a pagina 59, `createWebSocketInbound` è un'applicazione del *design pattern* Factory Method, in quanto la sottoclasse di `WebSocketServlet` restituisce un oggetto di tipo statico `StreamInbound` ma avente il tipo dinamico adatto agli scopi dell'applicazione.

Diagramma delle classi:

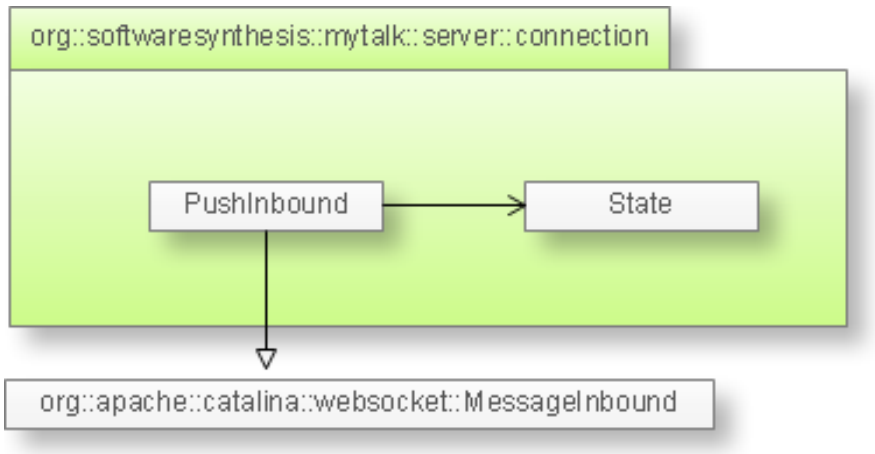

Figura 3: Diagramma delle classi - Gestione connessione

Classi utilizzate:

- connection.PushInbound
- connection.State
- org.apache.catalina.websocket.MessageInbound

6.1.3 CS03 – Gestione rubrica

Descrizione:

A ogni utente del sistema, che corrisponde a un'istanza di abook.UserData (implementazione dell'interfaccia abook.IUserData), è associata una rubrica personale.

La rubrica è rappresentata da una lista di abook.IAddressBookEntry, per la quale si fornisce un'implementazione nominata abook.AddressBookEntry che rappresenta un contatto della rubrica di un utente. Come si evince dal diagramma riportato in figura 4, la struttura del componente rispecchia la logica di funzionamento del database.

Di conseguenza ogni istanza di tipo abook.AddressBookEntry apparterrà a un solo abook.IUserData e in essa è registrata un'istanza di abook.UserData che rappresenta il contatto facente parte della rubrica.

Gli utenti inoltre possono essere opzionalmente organizzati in gruppi: un contatto in rubrica può, in un dato momento, appartenere a uno o più gruppi oppure non appartenere a nessuno. Per tale motivo ogni istanza di abook.AddressBookEntry contiene anche un riferimento di tipo IGroup.

Tale interfaccia, congiuntamente alla sua implementazione abook.Group, permettono di rappresentare i gruppi della rubrica di un utente.

Infine sono state predisposte 12 *servlet* per interagire con le classi qui descritte:

- abook.servlet.AddressBookDoAddContactServlet
- abook.servlet.AddressBookDoRemoveContactServlet
- abook.servlet.AddressBookDoCreateGroupServlet
- abook.servlet.AddressBookDoDeletGroupServlet
- abook.servlet.AddressBookDoInsertInGroupServlet
- abook.servlet.AddressBookDoRemoveInGroupServlet
- abook.servlet.AddressBookDoBlockServlet
- abook.servlet.AddressBookDoUnblockServlet
- abook.servlet.AddressBookGetContactsServlet
- abook.servlet.AddressBookGetGroupsServlet
- abook.servlet.AddressBookDoSearchServlet
- abook.servlet.AccountSettingsServlet

Le loro funzionalità riguardano principalmente le possibilità di modificare, creare e cancellare istanze di oggetti del tipo transfer object e quindi riconducibili ad oggetti fisicamente presenti nel database. Le *servlet* saranno descritte con maggiori dettaglio nella sezione 6.1.7 a pagina 22 (CS07 – Façade del server).

Diagramma delle classi:

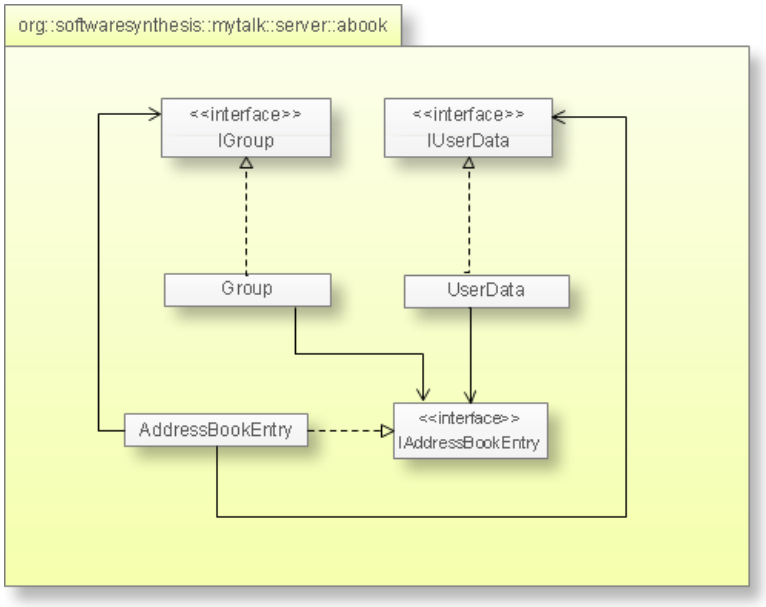

Figura 4: Diagramma delle classi - Gestione rubrica

Classi utilizzate:

- `abook.AddressBookEntry`
- `abook.IAddressBookEntry`
- `abook.IGroup`
- `abook.Group`
- `abook.IUserData`
- `abook.UserData`

6.1.4 CS04 – Gestione autenticazione

Descrizione:

L'autenticazione passa attraverso due fasi: la richiesta di *login* e l'approvazione (*commit*). La richiesta di *login* verifica le credenziali di accesso con i dati memorizzati all'interno del database, mentre la richiesta di *commit* assegna al *subject* che ha attivato la procedura di autenticazione, dei valori identificativi che permettano di identificarlo finché esso utilizza il sistema software.

Quando un utente avvia la procedura di autenticazione la *servlet* che espone le funzionalità di questo componente, `authentication.servlet.LoginServlet` (che estende a sua volta `javax.servlet.http.HttpServlet`) crea il contesto di autenticazione generando un istanza di `authentication.CredentialLoader` che ha il compito di caricare le credenziali di accesso e accede al file di configurazione per istanziare il giusto modulo per l'autenticazione, costituito dalla classe `authentication.AuthenticationModule`.

La componente predispone anche le funzionalità da usare per eseguire il *logout* di un utente registrato nel sistema, e la registrazione al sistema stesso. Tale funzionalità saranno richiamabili mediante 2 servlet:

- `authentication.servlet.LogoutServlet`
- `authentication.servlet.RegisterServlet`

Per garantire un buon livello di sicurezza nell'invio dei dati (`authentication.AuthenticationData`) il sistema si appoggia ad un sistema di criptazione basato su algoritmo AES. La classe che fornisce tale funzionalità è `authentication.AESAlgoritm`, implementazione dell'interfaccia `authentication.ISecurityStrategy`.

Diagramma delle classi:

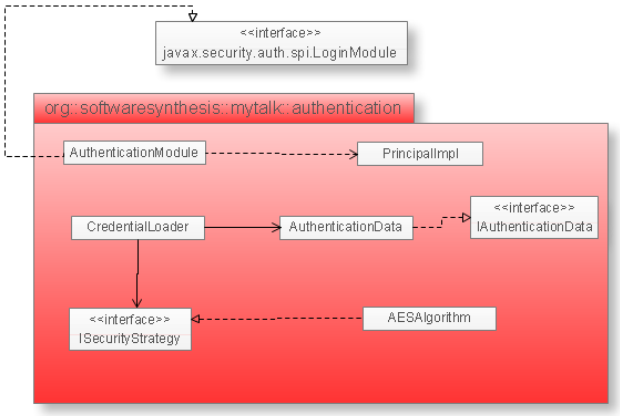

Figura 5: Diagramma delle classi - Gestione autenticazione

Classi utilizzate

- `authentication.AuthenticationModule`
- `authentication.CredentialLoader`

- authentication.PrincipalImpl
- authentication.IAuthenticationData
- authentication.AuthenticationData
- authentication.AESAlgorithm
- authentication.ISecurityStrategy
- javax.security.auth.spi.LoginModule
- javax.security.auth.callback.CallbackHandler
- javax.security.Principal

6.1.5 CS05 – Gestione segreteria

Descrizione:

I messaggi in segreteria, che possono essere di natura audio o audio/video, corrispondono alle istanze dalla classe `message.Message` (implementazione dell'interfaccia `message.IMessage`) e sono caratterizzati da un utente mittente, da un destinatario e dalla data di registrazione.

La collezione di messaggi aventi un determinato utente come destinatario può essere ottenuta tramite un'operazione dichiarata nell'interfaccia `abook.IUserData`.

Per consentire ai client di accedere alle funzionalità gestite da questo componente sono state predisposte delle *servlet* che, per loro natura, estendono `javax.servlet.http.HttpServlet` e che saranno descritte con maggiori dettagli nella sezione 6.1.7 a pagina 22. Al fine di comunicare le funzionalità proposte viene di seguito riportata la lista delle *servlet* che comunicano con tale componente:

- `message.servlet.InsertMessageServlet`: viene utilizzata da un client A per lasciare un messaggio nella segreteria di un client B;
- `message.servlet.DeletMessageServlet`: viene utilizzata da un client per rimuovere un messaggio presente nella propria segreteria;
- `message.servlet.UpdateStatusMessageServlet`: viene utilizzata per modificare lo stato di un messaggio da "da leggere" a "letto";
- `message.servlet.DownloadMessageListServlet`: viene utilizzata da un client per scaricare la lista dei messaggi presenti nella sua segreteria;

Diagramma delle classi:

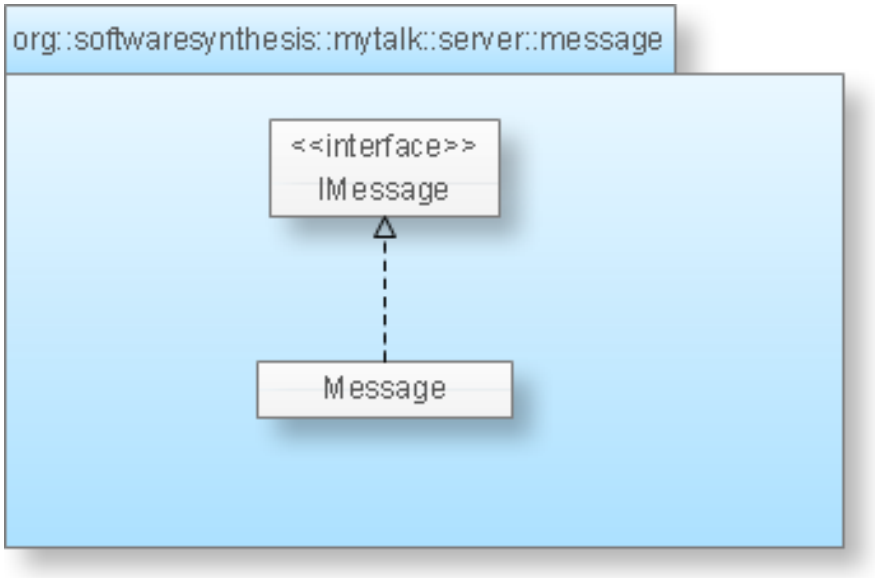

Figura 6: Diagramma delle classi - Gestione segreteria

Classi utilizzate:

- `message.IMessage`
- `message.Message`

6.1.6 CS06 – Gestione chiamate

Descrizione:

Le classi di questo componente hanno il ruolo di definire una strutturata per definire lo storico delle chiamate di un utente e, conseguentemente, l'accesso ai dati relativi a una chiamata (data, mittente e $n \geq 1$ destinatari della chiamata).

La componente contiene l'interfaccia `ICall` e la sua implementazione lato server `Call`, che rappresenta il *transfer object* per la rappresentazione delle chiamate che deve essere mappata nel database.

Infine, allo scopo di consentire ai client l'accesso alle funzionalità di questo componente, è stata prevista una *servlet* `call.servlet.DownloadCallHistoryServlet` che estende `javax.servlet.http.HttpServlet` il cui ruolo sarà descritto con maggiori dettagli nella sezione 6.1.7 nella pagina successiva.

Diagramma delle classi:

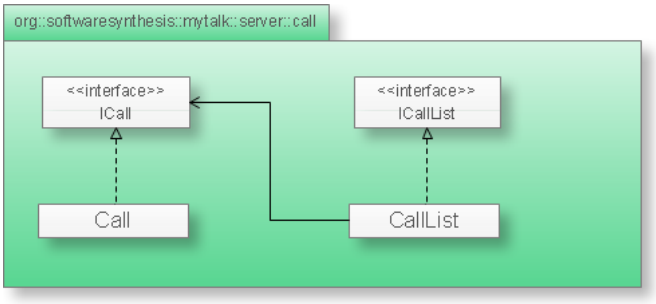

Figura 7: Diagramma delle classi - Gestione chiamate

Classi utilizzate:

- `call.ICall`
- `call.Call`
- `call.ICall`
- `call.Call`

6.1.7 CS07 – Façade del server

Descrizione:

Tale componente ha il ruolo di offrire ai client una serie di punti di accesso alle funzionalità del sistema server. Dal momento che tale interazione avviene in un contesto distribuito, i punti di accesso ai componenti del server sono stati implementati come *servlet* Java destinate a essere eseguite nell'ambiente del *servlet container* fornito da *Apache TomCat*.

Per questo motivo, tali Façade saranno realizzate come estensioni della classe di libreria `javax.servlet.http.HttpServlet`, al fine di creare un sistema di risposta lato server che prenderà in carico le richieste provenienti dai client.

La classe Java `HttpServlet` implementa un sistema di *listening* che permette di monitorare costantemente richieste in ingresso. Queste ultime possono essere di duplice natura e corrispondono ai metodi di invio dei dati previsti dal protocollo HTTP (`GET` e `POST`).

La mappatura fra componenti del server e le rispettive *servlet* di Façade, implementate come sottoclassi di `javax.servlet.http.HttpServlet`, è illustrata dalla tabella seguente:

Componente	Servlet di Façade
CS02 – Gestione connessione	connection.ChannelServlet
CS03 – Gestione rubrica	abook.servlet.AddressBookDoAddContactServlet
	abook.servlet.AddressBookDoRemoveContactServlet
	abook.servlet.AddressBookDoCreateGroupServlet
	abook.servlet.AddressBookDoDeletGroupServlet
	abook.servlet.AddressBookDoInsertInGroupServlet
	abook.servlet.AddressBookDoRemoveInGroupServlet
	abook.servlet.AddressBookDoBlockServlet
	abook.servlet.AddressBookDoUnblockServlet
	abook.servlet.AddressBookGetContactsServlet
	abook.servlet.AddressBookGetGroupsServlet
	abook.servlet.AddressBookDoSearchServlet
	abook.servlet.AccountSettingsServlet
CS04 – Gestione autenticazione	authentication.servlet.LoginServlet
	authentication.servlet.LogoutServlet
	authentication.servlet.RegisterServlet
CS05 – Gestione segreteria	message.servlet.InsertMessageServlet
	message.servlet.DeletMessageServlet
	message.servlet.UpdateStatusMessageServlet
	message.servlet.DownloadMessageListServlet
CS06 – Gestione chiamate	call.servlet.DownloadCallHistoryManager

In particolare `connection.ChannelServlet` è coinvolto dopo la fase di *login*, una volta che è stata accertata la validità dell'autenticazione, interagisce con Gestione connessione allo scopo di creare un canale di comunicazione fra il server e il client, secondo le modalità illustrate nella sezione 6.1.2.[2]

Il compito di interfacciarsi con Gestione rubrica per recuperare la rubrica associata ad un utente e per riflettere sul server le operazioni di amministrazione della rubrica spetta invece al pacchetto di *servlet* `abook.servlet`. Tale *servlet* risponde anche alle richieste dei client per le azioni di aggiornamento e modifica dell'account personale dell'utente associato al client. Nello specifico si ha:

- `abook.servlet.AddressBookDoAddContactServlet`: richiamata da un client per aggiungere un contatto alla propria rubrica utente;
- `abook.servlet.AddressBookDoRemoveContactServlet`: richiamata da un client per rimuovere un contatto dalla propria rubrica;
- `abook.servlet.AddressBookDoCreateGroupServlet`: richiamata da un client per creare un nuovo gruppo contatti;
- `abook.servlet.AddressBookDoDeletGroupServlet`: richiamata da un client per eliminare un gruppo;
- `abook.servlet.AddressBookDoInsertInGroupServlet`: richiamata da un client per inserire un contatto in un gruppo;
- `abook.servlet.AddressBookDoRemoveInGroupServlet`: richiamata da un client per rimuovere un contatto da un gruppo della propria rubrica;
- `abook.servlet.AddressBookDoBlockServlet`: richiamata da un client per bloccare un contatto presente nella propria rubrica, indifferentemente dai gruppi (della rubrica del client) in cui esso si trova;

[2]Tale classe, come anticipato nella descrizione a pagina 16, non è in realtà sottotipo diretto di `javax.servlet.http.HttpServlet`, ma estende `org.apache.catalina.websocket.WebSocketServlet` che a sua volta è sottoclasse di `javax.servlet.http.HttpServlet`.

- abook.servlet.AddressBookDoUnblockServlet: richiamata da un client per sbloccare un contatto presente nella propria rubrica;
- abook.servlet.AddressBookGetContactsServlet: richiamata da un client per scaricare la lista dei contatti presenti nella propria rubrica;
- abook.servlet.AddressBookGetGroupsServlet: richiamata da un client per scaricare lista dei contatti e dei gruppi in cui essi sono inseriti;
- abook.servlet.AddressBookDoSearchServlet: utilizzata per ricercare gli utente contenenti nei parametri nome, cognome o mail, una parola chiave usata per fare la ricerca e passata alla *servlet*;
- abook.servlet.AccountSettingsServlet: richiamata da un client per modificare i propri dati personali memorizzati nel server.

I cambiamenti di stato di un utente sono invece processati da connection.ChannelServlet che fornisce di conseguenza anch'esso un punto d'accesso al componente Gestione rubrica.

Ai fini del *login* è utilizzata invece la *servlet* authentication.servlet.LoginServlet, che ha il compito di interfacciarsi con il sistema di autenticazione e, in particolare, con l'istanza del sottotipo di javax.security.auth.spi.LoginModule in uso per verificare le credenziali. Inoltre come già accennato sono presenti la *servlet* per il *logout* authentication.servlet.LogoutServlet e quella per la registrazione al sistema authentication.servlet.RegisterServlet.

Infine lo scaricamento e la gestione della segreteria telefonica (di competenza di Gestione segreteria) e dello storico delle chiamate (per quanto riguarda il componente Gestione chiamate) avvengono in risposta alle richieste dei client pervenute al pacchetto di servlet message.servlet e call.servlet.DownloadCallHistoryServlet rispettivamente.

Diagramma delle classi:

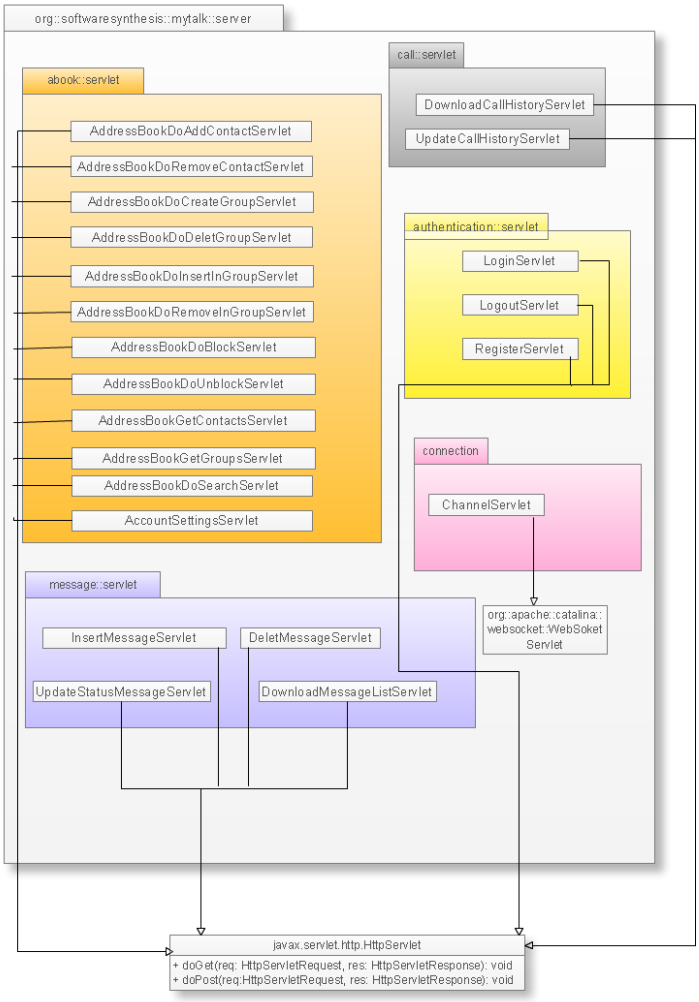

Figura 8: Diagramma delle classi - Façade del server

Il diagramma in figura 8 rappresenta la gerarchia delle *servlet* di Façade ma non mostra in dettaglio le dipendenze fra ognuna di queste e i componenti interni ai sotto-package `abook`, `authentication`, `call`, `connection` e `message` per non appesantire troppo la notazione.

Tali dipendenze, di fondamentale importanza per l'architettura, sono però riportate nei diagrammi della sezione 9.2 a pagina 55 in cui è descritta con maggiori dettagli l'applicazione del *design pattern* Façade.

Classi utilizzate:

- `connection.ChannelServlet`
- `abook.servlet.AddressBookDoAddContactServlet`
- `abook.servlet.AddressBookDoRemoveContactServlet`
- `abook.servlet.AddressBookDoCreateGroupServlet`
- `abook.servlet.AddressBookDoDeletGroupServlet`

- abook.servlet.AddressBookDoInsertInGroupServlet
- abook.servlet.AddressBookDoRemoveInGroupServlet
- abook.servlet.AddressBookDoBlockServlet
- abook.servlet.AddressBookDoUnblockServlet
- abook.servlet.AddressBookGetContactsServlet
- abook.servlet.AddressBookGetGroupsServlet
- abook.servlet.AddressBookDoSearchServlet
- abook.servlet.AccountSettingaServlet
- authentication.servlet.LoginServlet
- authentication.servlet.LogoutServlet
- authentication.servlet.RegisterServlet
- message.servlet.InsertMessageServlet
- message.servlet.DeletMessageServlet
- message.servlet.UpdateStatusMessageServlet
- message.servlet.DownloadMessageListServlet
- call.servlet.DownloadCallHistoryManager
- javax.servlet.http.HttpServlet
- org.apache.catalina.websocket.WebSocketServlet

6.2 Descrizione delle classi

6.2.1 Package `org.softwaresynthesis.mytalk.server.dao`

- `AddressBookEntryDAO`

  Descrizione:
  Tale classe espone la funzione per la manipolazione di oggetti di tipo `IAddressBookEntry` con il database.

  Componenti che ne fanno uso:
  - CS01 – Gestione database
  - CS07 – Façade del server

- `CallDAO`

  Descrizione:
  Tale classe interagisce con il *framework* Hibernate e contiene la logica per la manipolazione dei dati (in lettura e scrittura) nella tabella *Calls* del database.

  Componenti che ne fanno uso:
  - CS01 – Gestione database
  - CS07 – Façade del server

- `CallListDAO`

  Descrizione:
  Tale classe interagisce con il *framework* Hibernate e contiene la logica per la manipolazione dei dati (in lettura e scrittura) nella tabella *CallLists* del database che mappa la relazione molti a molti tra *UserData* e *Calls*.

  Componenti che ne fanno uso:
  - CS01 – Gestione database
  - CS07 – Façade del server

- `MessageDAO`

Descrizione:

Tale classe interagisce con il *framework* Hibernate e contiene la logica per la manipolazione dei dati (in lettura e scrittura) nella tabella *Message* del database.

Componenti che ne fanno uso:

- CS01 – Gestione database
- CS07 – Façade del server

- `GroupDAO`

Descrizione:

Tale classe interagisce con il *framework* Hibernate e contiene la logica per la manipolazione dei dati (in lettura e scrittura) nella tabella *Group* del database.

Componenti che ne fanno uso:

- CS01 – Gestione database
- CS07 – Façade del server

- `UserDataDAO`

Descrizione:

Tale classe interagisce con il *framework* Hibernate e contiene la logica per la manipolazione dei dati (in lettura e scrittura) nella tabella *UserData* del database.

Componenti che ne fanno uso:

- CS01 – Gestione database
- CS04 – Gestione autenticazione
- CS07 – Façade del server

- `HibernateUtil`

Descrizione:
Classe che implementa il *pattern* Singleton e che viene utilizzata dalle classi DAO al fine di ottenere un riferimento alla sessione di connessione al DBMS tramite la quale effettuare le transazioni corrispondenti alle operazioni volte ad assicurare la persistenza dei dati.

Componenti che ne fanno uso:

- CS01 – Gestione database
- CS07 – Façade del server

6.2.2 Package `org.softwaresynthesis.mytalk.server.connection`

- `PushInbound`

Descrizione:
Sottoclasse di `org.apache.catalina.websocket.MessageInbound` (e, di conseguenza sottotipo anche di `org.apache.catalina.websocket.StreamInbound`) che rappresenta il canale di comunicazione bidirezionale fra client e server realizzato mediante *WebSocket*.

Il comportamento della sotto-architettura server in risposta ai messaggi ricevuti dal client è completamente determinato dall'implementazione (*overriding*) che tale classe fornisce del metodo `void onTextMessage(CharBuffer)`.

Componenti che ne fanno uso:

- CS02 – Gestione connessione

- CS07 – Façade del server

- `State`

  Descrizione:
  Classe che attraverso un tipo enumerativo rappresenta lo stato in cui si trova un utente. Si osservi che tale stato non è associato direttamente all'utente ma bensì al canale di comunicazione (univoco) di cui l'utente diventa proprietario al momento del *login*. Di conseguenza le istanze della classe `connection.PushInBound` sono composte con un'istanza di `abook.State` che rappresenta lo stato in cui si trova un determinato utente in un determinato momento.

  Componenti che ne fanno uso:
  - CS02 – Gestione rubrica
  - CS07 – Façade del server

- `ChannnelServlet`

  Descrizione:
  Sottoclasse di `org.apache.catalina.websocket.WebSocketServlet` che ha la funzione di punto di accesso al componente Gestione connessione grazie al quale, ad autenticazione avvenuta, viene instaurato un canale di comunicazione fra il server e il client destinato a essere usato per effettuare e ricevere le comunicazioni fra client.
  Come accennato nella sezione 6.1.2 a pagina 16, tramite un opportuno *overriding* del metodo `createWebSocketInbound(String, HttpServletRequest`, la *servlet* restituisce ai client un canale di comunicazione che, per la sua natura *full-duplex*, può essere utilizzato sia dal client per trasmettere messaggi al server che viceversa.
  Questo Factory Method (per la cui descrizione si rimanda alla sezione 9.3) è richiamato dal *servlet container* nel momento in cui lato client, ad autenticazione ultimata, viene invocato il costruttore di `WebSocket` passando a quest'ultimo come parametro l'URL della *servlet* `ChannelServlet`.
  In ultima, la *servlet* si occupa di notificare i cambiamenti di stato degli utenti presenti nella rubrica di ogni altro utente.

  Componenti che ne fanno uso:
  - CS02 – Gestione connessione
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

6.2.3 Package `org.softwaresynthesis.mytalk.server.abook`

- `IAddressBookEntry`

  Descrizione:
  Interfaccia che raccoglie le operazioni per la gestione di un utente appartenente alla rubrica di un altro utente (metodi *get*/*set* per accedere ai dati del contatto).

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookEntry`

  Descrizione:
  Classe che implementa l'interfaccia `IAddressBookEntry` e rappresenta quindi un contatto della rubrica associata a un determinato utente.
  Permette di accedere al contatto, al suo gruppo e di determinare se tale utente risulta essere bloccato dall'utente possessore del contatto.

Componenti che ne fanno uso:

- CS03 – Gestione rubrica
- CS07 – Façade del server

- `IUserData`

  Descrizione:
  Interfaccia per le classi che rappresentano gli utenti, è dotata di operazioni *get*/*set* per accedere ai dati degli utenti registrati sul sistema.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `UserData`

  Descrizione:
  Classe *transfer object* che implementa l'interfaccia `IUserData` le cui istanze corrispondono ai *record* della tabella degli utenti nel database.

  È caratterizzata dai campi dati che corrispondono alle proprietà degli utenti (email, password, domanda segreta e relativa risposta, nome, cognome e immagine personale).

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

- `IGroup`

  Descrizione:
  Interfaccia per i gruppi interni alla rubrica di un utente, prevede un'operazione astratta `add(IUserData)` per l'aggiunta di un nuovo contatto al gruppo e `remove(IUserData)` per la sua rimozione di un contatto dal gruppo. Contiene anche i metodi *get*/*set* per impostare o recuperare il nome di un gruppo.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `Group`

  Descrizione:
  Implementazione dell'interfaccia `IGroup` che costituisce anche il *transfer object* per i gruppi di una rubrica utente. Ogni gruppo è dotato di un nome e raccoglie in sé uno o più istanze di classi sottotipo di `abook.IUserData`.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

6.2.4 Package `org.softwaresynthesis.mytalk.server.abook.servlet`

- `AddressBookDoAddContactServlet`

  Descrizione:
  Servlet il cui compito consiste nell'aggiungere alla rubrica personale dell'utente un nuovo contatto tra quelli registrati al sistema MyTalk.

Componenti che ne fanno uso:
- CS03 – Gestione rubrica
- CS07 – Façade del server

- `AddressBookDoRemoveContactServlet`

  Descrizione:
  Servlet il cui compito consiste nel rimuovere dalla rubrica personale dell'utente un contatto tra quelli presenti nella rubrica stessa.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookDoBlockServlet`

  Descrizione:
  Servlet il cui compito consiste nel bloccare un contatto presente nella rubrica dell'utente, in questo modo lo stesso non può essere più contattato dal contatto.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookDoUnblockServlet`

  Descrizione:
  Servlet il cui compito consiste nello sbloccare un contatto presente nella rubrica dell'utente, in questo modo lo stesso può essere contattato dal contatto che era stato in precedenza bloccato.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookDoCreateGroupServlet`

  Descrizione:
  Servlet il cui compito consiste inserire un nuovo gruppo in cui inserire gli utenti presenti nella rubrica personale dell'utente.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookDoDeleteGroupServlet`

  Descrizione:
  Servlet il cui compito consiste eliminare un gruppo creato in precedenza e quindi presente nella rubrica personale dell'utente.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookDoInsertInGroupServlet`

  Descrizione:
  Servlet il cui compito consiste nell'inserire un contatto presente nella rubrica dell'utente ad un gruppo esistente nella rubrica personale.

Componenti che ne fanno uso:
- CS03 – Gestione rubrica
- CS07 – Façade del server

- `AddressBookDoRemoveFromGroupServlet`

  Descrizione:
  Servlet il cui compito consiste rimuovere un contatto presente nella rubrica dell'utente da un gruppo esistente nella rubrica personale.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AddressBookGetContactsServlet`

  Descrizione:
  Servlet il cui compito consiste nel fornire la rubrica di un utente iscritto al sistema MyTalk.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

- `AccountSettingsServlet`

  Descrizione: *Servlet* che ha il compito di permettere la modifica dei dati personali di un utente iscritto al sistema MyTalk.

  Componenti che ne fanno uso:
  - CS03 – Gestione rubrica
  - CS07 – Façade del server

6.2.5 Package `org.softwaresynthesis.mytalk.server.message`

- `IMessage`

  Descrizione:
  Interfaccia che dichiara le operazioni astratte (*get*/*set*) necessarie ad accedere alle proprietà di un messaggio in segreteria (in particolare: mittente, destinatario, stato del messaggio e data di registrazione).

  Componenti che ne fanno uso:
  - CS05 – Gestione segreteria
  - CS07 – Façade del server

- `Message`

  Descrizione:
  Classe *transfer object* che rappresenta un messaggio di natura audio o audio/video nella segreteria telefonica di un utente e implementa l'interfaccia `IMessage`.

  Componenti che ne fanno uso:
  - CS05 – Gestione segreteria
  - CS07 – Façade del server

6.2.6 Package org.softwaresynthesis.mytalk.server.message.servlet

- InsertMessageServlet

  Descrizione:
  Sottoclasse di javax.servlet.http.HttpServlet che rappresenta un punto di accesso al componente Gestione segreteria e nello specifico consente di inserire un nuovo messaggio nella segreteria di un utente del sistema.

  Componenti che ne fanno uso:
  - CS07 – Façade del server
  - CP03 – Gestione GUI

- UpdateStatusMessageServlet

  Descrizione:
  Sottoclasse di javax.servlet.http.HttpServlet che rappresenta un punto di accesso al componente Gestione segreteria e nello specifico consente di modificare lo stato di un messaggio presente nella segreteria utente.

  Componenti che ne fanno uso:
  - CS07 – Façade del server
  - CP03 – Gestione GUI

- DeleteMessageServlet

  Descrizione:
  Sottoclasse di javax.servlet.http.HttpServlet che rappresenta un punto di accesso al componente Gestione segreteria e nello specifico consente l'eliminazione di un messaggio presente nella segreteria del client che richiede l'operazione di cancellazione.

  Componenti che ne fanno uso:
  - CS07 – Façade del server
  - CP03 – Gestione GUI

- DownloadMessageListServlet

  Descrizione:
  Sottoclasse di javax.servlet.http.HttpServlet che rappresenta un punto di accesso al componente Gestione segreteria e nello specifico permette di scaricare la lista dei messaggi.

  Componenti che ne fanno uso:
  - CS07 – Façade del server
  - CP03 – Gestione GUI

6.2.7 Package org.softwaresynthesis.mytalk.server.call

- ICall

  Descrizione:
  Interfaccia che contiene le operazioni astratte di accesso ai dati (metodi *get*/*set*) che caratterizzano le chiamate, vale a dire il chiamante, l'insieme degli utenti chiamati nonché la data e l'ora di avvio e di terminazione della chiamata.

  Componenti che ne fanno uso:
  - CS06 – Gestione chiamate
  - CS07 – Façade del server

- Call

Descrizione:
Classe *transfer object* che rappresenta le chiamate realizzate attraverso il sistema e implementa l'interfaccia `ICall`.

Componenti che ne fanno uso:
- CS06 – Gestione chiamate

- `ICallList`

Descrizione:
Interfaccia che contiene le operazioni astratte di accesso ai dati (metodi *get*/*set*) che caratterizzano una voce nella lista delle chiamate.

Componenti che ne fanno uso:
- CS06 – Gestione chiamate

- `CallList`

Descrizione:
Classe *transfer object* che rappresenta una voce nella lista delle chiamate realizzate attraverso il sistema e implementa l'interfaccia `ICallList`.

Componenti che ne fanno uso:
- CS06 – Gestione chiamate

6.2.8 Package `org.softwaresynthesis.mytalk.server.call.servlet`

- `DownloadCallHistoryServlet`

Descrizione:
Sottoclasse di `javax.servlet.http.HttpServlet` la cui responsabilità è intercettare le richieste aventi come oggetto lo storico delle chiamate utente, che deve essere restituito dal server al client in forma opportunamente serializzata per l'interscambio di dati. Non è previsto invece alcun tipo di operazione "attiva" sui dati memorizzati nel server.

Componenti che ne fanno uso:
- CS07 – Façade del server
- CP03 – Gestione GUI

6.2.9 Package `org.softwaresynthesis.mytalk.server.authentication`

- `AuthenticationModule`

Descrizione
Implementazione dell'interfaccia `javax.security.auth.spi.LoginModule` definisce la logica di autenticazione attuata dal sistema MyTalk. Se si avesse bisogno di una maggiore sicurezza per questa procedura è sufficiente fornire una nuova implementazione di questa interfaccia.

Componenti che ne fanno uso
- CS04 – Gestione autenticazione
- CS07 – Façade del server

- `CredentialLoader`

Descrizione
La classe ha compito di caricare le credenziali di accesso, fornite dall'utente, con lo scopo di metterle a disposizione del modulo di autenticazione `AuthenticationModule`.

Componenti che ne fanno uso
- CS04 – Gestione autenticazione

- CS07 – Façade del server

- `PrincipalImpl`

  Descrizione
  La classe serve per contenere un elemento identificativo dell'utente che ha effettuato l'autenticazione, in modo che possa essere facilmente identificato nelle altre componenti del software.

  Componenti che ne fanno uso
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

- `IAuthenticationData`

  Descrizione
  Interfaccia che contiene le credenziali fornite in *input* dall'utente per avviare successivamente la procedura di *login* al sistema MyTalk.

  Componenti che ne fanno uso
  ,nolistsep]
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

- `AuthenticationData`

  Descrizione
  La classe contiene le credenziali di accesso, fornite dall'utente, ed utilizzate da `CredentialLoader`.

  Componenti che ne fanno uso
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

- `ISecurityStrategy`

  Descrizione
  Interfaccia che contiene la strategia di crittografia adottata dal sistema MyTalkper garantire la sicurezza dei dati dell'utente registrato al sistema stesso.

  Componenti che ne fanno uso
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

- `AESAlgorithm`

  Descrizione
  Implementa l'interfaccia `ISecurityStrategy` implementando la strategia di cripting, ovvero adottando l'algoritmo AES a 128 bit scelto dal team.

  Componenti che ne fanno uso
  - CS04 – Gestione autenticazione
  - CS07 – Façade del server

6.2.10 Package `org.softwaresynthesis.mytalk.server.authentication.servlet`

- `LoginServlet`

  Descrizione
  Servlet che ha il compito di autorizzare l'accesso al sistema MyTalkdi un determinato utente precedentemente iscritto.

  Componenti che ne fanno uso
  - CS04 – Gestione autenticazione

- `LogoutServlet`

  Descrizione

  Servlet che ha il compito di terminare l'accesso al sistema MyTalkda parte di un determinato utente precedentemente autenticato.

  Componenti che ne fanno uso

  - CS04 – Gestione autenticazione

- `RegisterServlet`

  Descrizione

  Servlet usata per garantire la registrazione di un utente al sistema MyTalk.

  Componenti che ne fanno uso

  - CS04 – Gestione autenticazione

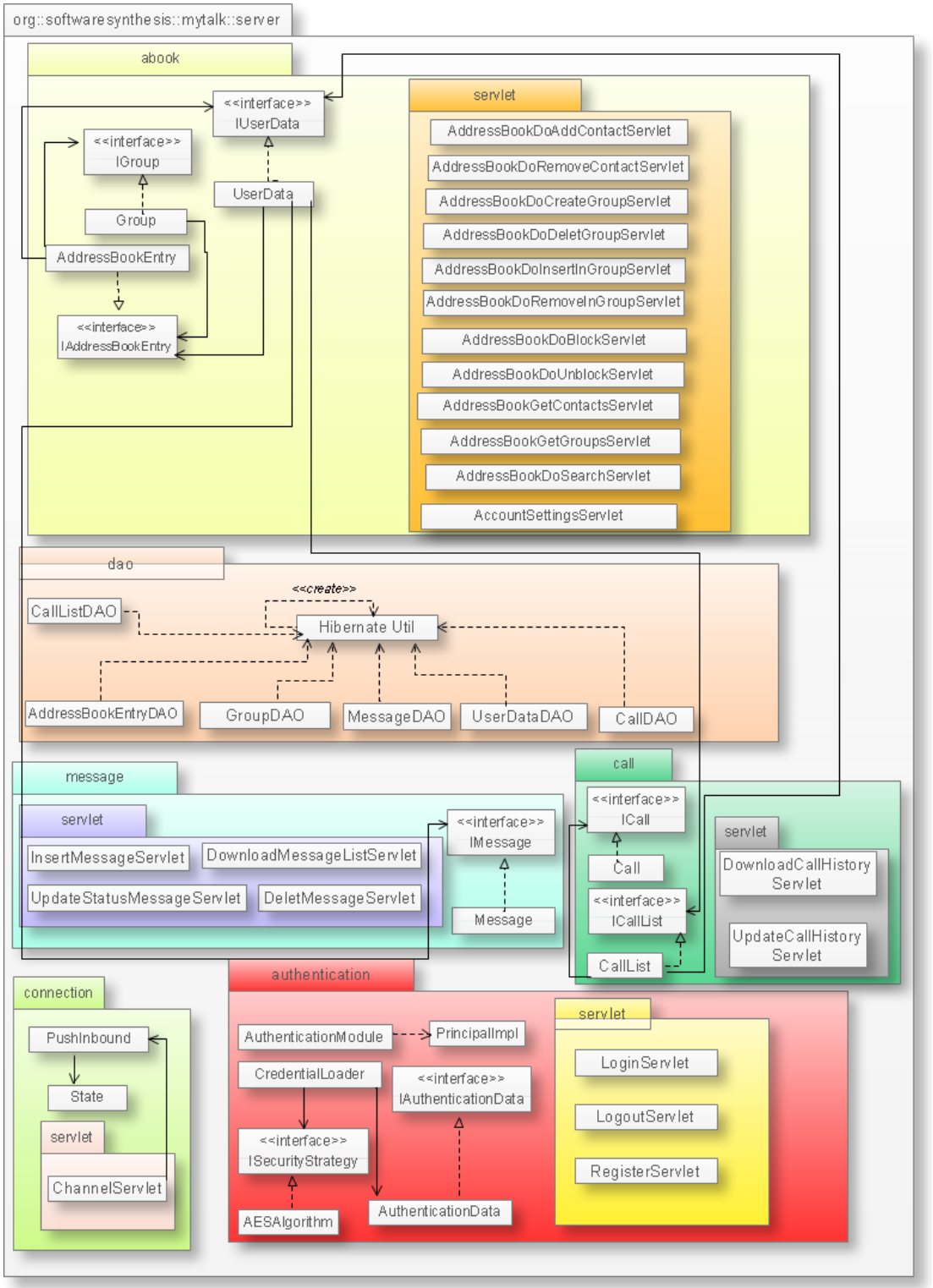

Figura 9: Diagramma delle classi - Architettura mytalk.server

Il diagramma in figura 9 rappresenta solo ed esclusivamente le classi definite in sede di progettazione architetturale e non riporta invece tutte le interfacce/classi appartenenti a librerie esterne (ad es. `javax.servlet.http.HttpServlet`) che sono state estese o implementate nella sotto-architettura server. Tali informazioni, che avrebbero appesantito ulteriormente un diagramma già di grandi dimensioni sono invece riportate nei diagrammi dei singoli componenti. Inoltre il diagramma non riporta le dipendenze.

7 Architettura mytalk.clientpresenter

La sotto-architettura **clientpresenter**, nasce con lo scopo di imporre una separazione tra la logica di gestione di un client e l'interfaccia grafica visualizzata all'utente finale.

Dal momento che il *presenter* ha la responsabilità di ricevere i comandi utente dalla vista e di sovrintendere all'aggiornamento della stessa, è prevista anche la presenza di una serie di oggetti necessari a gestire l'interazione con la vista, corrispondenti al componente Gestione GUI.

Dal momento che per l'interazione con l'utente sono previste molteplici viste, è stato predisposto un *presenter* per ognuna di queste. Ogni vista detiene un riferimento al proprio *presenter* al quale vengono inviate le richieste su risposta dell'interazione con l'utente, in quanto le operazioni del *presenter* sono impostate come funzioni di *callback* nell'interfaccia grafica.

I vari *presenter* interagiscono con le viste grazie alla conoscenza di una "interfaccia" per la loro manipolazione, in quanto tramite *JavaScript* è possibile intervenire sul DOM. Tali considerazioni sono conformi all'utilizzo del *design pattern* MVP.[3]

Questo componente ha inoltre il compito di recuperare i dati presenti sul server relativi alla rubrica di un utente, la segreteria telefonica e lo storico delle chiamate che lo interessano. I diversi *presenter* del componente Gestione GUI hanno dunque anche il compito di interrogare il server al fine di procurarsi le informazioni con cui popolare l'interfaccia grafica utente.

Le informazioni sono rappresentate dalle strutture del componente Rappresentazione dati e permangono internamente al *presenter* per essere disponibili a successive interrogazioni da parte della GUI senza coinvolgere nuovamente il server.

La logica di comunicazione fra client, e fra client e server, è incapsulata all'interno delle classi facenti parte del componente Gestione comunicazione. Queste ultime sfruttano un canale di comunicazione bidirezionale client-server per interagire con il server in merito alle richieste di comunicazione con altri client, andando a creare appositi canali di comunicazione che si avvalgono della tecnologia WebRTC.

In sintesi, la sotto-architettura **clientpresenter** dispone dei seguenti componenti:

- CP01 – Gestione comunicazione;
- CP02 – Rappresentazione dati;
- CP03 – Gestione GUI;

che saranno descritte con maggiori dettagli nella sezione successiva.

Si ricorda sin da ora che i nomi di tutte le interfacce e tutte le classi riportate nella sezione sono implicitamente parte del package `org.softwaresynthesis.mytalk.clientpresenter` pertanto tale prefisso sarà omesso nella loro denominazione.

Nella progettazione ad alto livello è stato scelto di non tenere in considerazione il dominio tecnologico dell'applicativo, che non prevede la presenza di classi e interfacce. La progettazione sarà pertanto influenzata unicamente dalle specifiche e dai principi della programmazione orientata agli oggetti.

Tale scelta è motivata dalle seguenti considerazioni:

- l'utilizzo dei *pattern* architetturali necessita a monte di uno stile di progettazione ad oggetti;
- così facendo è garantita la possibilità di riuso del risultato progettazione architetturale sotto altri domini tecnologici.

La presenza delle interfacce, delle relazioni di implementazione e l'estensione fra classi saranno pertanto mantenute.

[3] Per una descrizione più dettagliata del *pattern* MVP si rimanda alla sezione 9.5.

7.1 Componenti evidenziati

7.1.1 CP01 – Gestione comunicazione

Descrizione:

È il componente che rappresenta il nucleo logico fondamentale della comunicazione. In esso è presente la classe `kernel.CommunicationCenter` che ha la responsabilità di tener traccia delle comunicazioni aperte con i client usando la tecnologia WebRTC. Per compiere il proprio lavoro, tale classe si avvale delle funzionalità fornite dal webRTC per creare e gestire la comunicazione. Nello specifico vengono usate le classi:

- PeerICECandidate;
- WebKitRTCPeerConnection;
- PeerSessionDescription;

Diagramma delle classi:

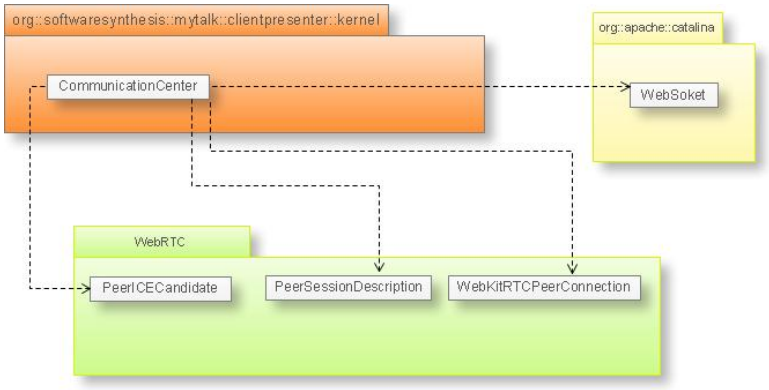

Figura 10: Diagramma delle classi - Gestione comunicazione

Classi utilizzate:

- `kernel.CommunicationCenter`
- `PeerICECandidate`
- `WebKitRTCPeerConnection`
- `PeerSessionDescription`

7.1.2 CP02 – Rappresentazione dati

Descrizione:

Le classi di questo componente hanno il compito di rappresentare sotto forma di apposite strutture dati le informazioni ottenute dall'interrogazione del modello dei dati e ricevute in forma serializzata a seguito delle richieste al server.

La disponibilità di queste informazioni sul client permette una più agevole rappresentazione dell'interfaccia grafica per gli oggetti del componente Gestione GUI ed evita nuove interrogazioni al server qualora i dati in possesso da parte del *presenter* siano ancora validi.

I dati rappresentati in forma di oggetti sono necessari, in particolare, per la gestione della segreteria, dello storico delle chiamate e della rubrica lato client.

Diagramma delle classi:

Figura 11: Diagramma delle classi - Rappresentazione dati

Classi utilizzate:

- `data.JSCall`
- `data.JSGroup`
- `data.JSMessage`
- `data.JSUserData`

7.1.3 CP03 – Gestione GUI

Descrizione:

Il componente ha il duplice ruolo di intercettare gli eventi che scaturiscono dall'interazione dell'utente con la GUI e di controllare l'aggiornamento della stessa a seguito dei cambiamenti che intervengono sul modello dei dati.

Per quanto riguarda l'autenticazione, è prevista la presenza di un *presenter* per l'interfaccia di *login*, corrispondente all'oggetto `guicontrol.LoginPanelPresenter`. Anche per la registrazione vale la stessa considerazione e tale funzionalità è destinata alla classe `guicontrol.RegisterPanelPresenter`

Per quanto riguarda invece la pagina principale dell'applicativo, alla quale si accede dopo aver effettuato l'autenticazione, è prevista la presenza di tre tipi principali di *presenter*:

- `guicontrol.AddressBookPanelPresenter`, che gestisce il pannello della rubrica presente nell'interfaccia grafica;
- `guicontrol.MainPanelPresenter`, alla radice di una gerarchia di *presenter* destinati a gestire gli elementi che compaiono nella parte centrale dell'interfaccia grafica, comprende alcune operazioni comuni a questi elementi grafici (ad es. per l'inizializzazione o per la comparsa/scomparsa dell'elemento grafico associato);
- `guicontrol.ToolsPanelPresenter`, che ha il compito di gestire il pannello degli strumenti nell'interfaccia grafica.

La struttura delle classi che lo compongono mostra una corrispondenza biunivoca con gli elementi grafici presentati nella sezione 8 con un *presenter* per ognuno dei pannelli cui si aggiunge un oggetto istanza di `guicontrol.PresenterMediator`.

Ogni *presenter* facente parte di questo componente contiene al suo interno le funzioni di *callback* che vengono richiamate dall'elemento dell'interfaccia grafica corrispondente.

Dal momento che il componente Gestione GUI è anche responsabile anche delle modifiche alle viste, le sottoclassi di `guicontrol.MainPanelPresenter` contengono anche le operazioni richiamate dall'esterno per l'aggiornamento dinamico della GUI.

L'oggetto `PresenterMediator` invece incapsula la collaborazione fra i vari *presenter* associati alla grafica e permette di gestire gli eventi generati da un elemento della GUI inviando a uno (o più) *presenter* i messaggi corrispondenti alle azioni da compiere.

Diagramma delle classi:

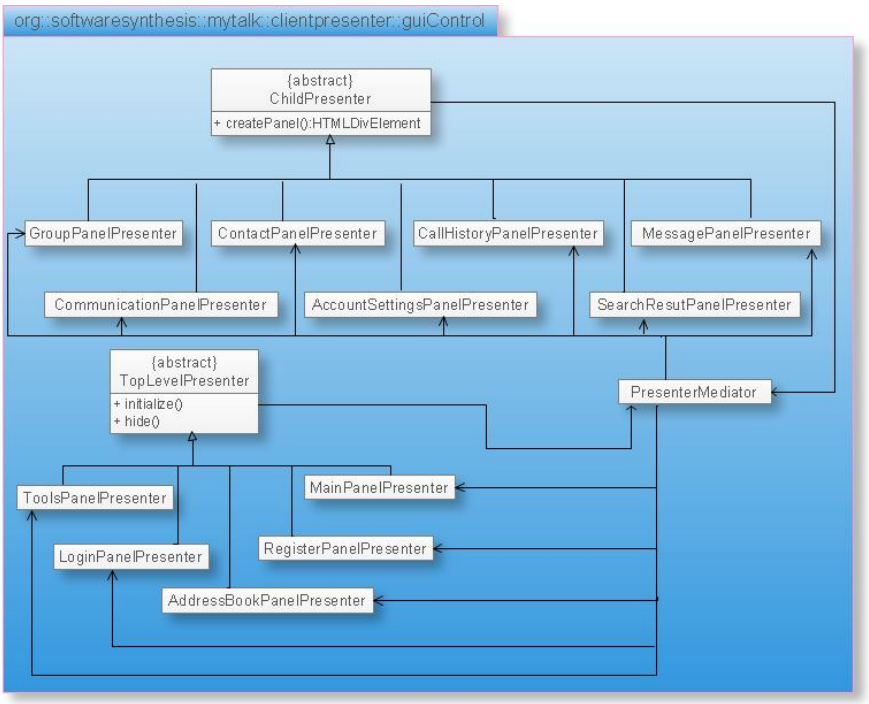

Figura 12: Diagramma delle classi - Gestione GUI

Classi utilizzate:

- guicontrol.AccountSettingsPanelPresenter
- guicontrol.AddressBookPanelPresenter
- guicontrol.CallHistoryPanelPresenter
- guicontrol.CommunicationPanelPresenter
- guicontrol.GroupPanelPresenter
- guicontrol.SearchResultPanelPresenter
- guicontrol.ContactPanelPresenter
- guicontrol.LoginPanelPresenter
- guicontrol.RegisterPanelPresenter
- guicontrol.TopLevelPresenter
- guicontrol.ChildPrenter
- guicontrol.MainPanelPresenter
- guicontrol.MessagePanelPresenter
- guicontrol.PresenterMediator
- guicontrol.ToolsPanelPresenter

7.2 Descrizione delle classi

7.2.1 Package org.softwaresynthesis.mytalk.clientpresenter.kernel

- CommunicationCenter

Descrizione:

È la classe che ha il compito di gestire la comunicazione con il server reagendo agli eventi che si verificano alla ricezione di un messaggio tramite il canale `server.connection.IChannel` (nel caso di una comunicazione entrante) oppure inviando tali messaggi al server (nel caso di una comunicazione uscente), al fine di stabilire una comunicazione client-client di tipo WebRTC.

Tale classe è inoltre stata implementata come Singleton, al fine di porre un limite superiore pari a uno al numero di istanze che sono presenti in ciascun client.

Componenti che ne fanno uso:

- CP01 – Gestione comunicazione

7.2.2 Package `org.softwaresynthesis.mytalk.clientpresenter.data`

- `JSCall`

Descrizione:

Tale classe modella una chiamata appartenente allo storico dell'utente corrente (dove risiede il *presenter*), e permette di accedere alle informazioni caratteristiche quali il mittente, l'insieme dei destinatari, la durata e la data di inizio della chiamata, sia essa di natura audio oppure audio/video.

Qualora lo storico delle chiamate non fosse disponibile sul client in cui è ospitato il *presenter*, gli oggetti di tipo `JSCall` sono creati a seguito di una richiesta da parte di `guicontrol.CallHistoryPanelPresenter` al fine di ottenere i dati che devono essere visualizzati nell'interfaccia grafica.

Componenti che ne fanno uso:

- CP02 – Rappresentazione dati
- CP03 – Gestione GUI

- `JSMessage`

Descrizione:

Classe che rappresenta un messaggio presente nella segreteria dell'utente corrente.

I diversi *presenter* del componente Gestione GUI possono accedere al mittente del messaggio, allo stato del messaggio (ascoltato o non ascoltato), alla natura del messaggio (audio o audio/video), alla durata e, in ultima istanza, al contenuto del messaggio.

I messaggi della segreteria telefonica di un determinato utente vengono scaricati dal *presenter* `guicontrol.MessagePanelPresenter` qualora non fossero presenti sul client che ospita il *presenter* e sono in seguito utilizzati per popolare l'interfaccia grafica.

Componenti che ne fanno uso:

- CP02 – Rappresentazione dati
- CP03 – Gestione GUI

- `JSUserData`

Descrizione:

Tale classe ha il compito di rappresentare i contatti della rubrica lato *presenter*, generata serializzando esclusivamente la parte che si vuole trasmettere ai client delle istanze del *transfer object* descritto sopra (`org.softwaresynthesis.mytalk.server.abook.UserData`).

In particolare, di un utente-contatto è possibile ottenere lo *username*, i dati anagrafici (nome e cognome) nonché l'immagine associata al profilo personale, laddove queste ultime tre informazioni sono da considerarsi facoltative e potrebbero non essere disponibili per un determinato utente.

Gli oggetti `JSUserData` vengono creati a seguito di una richiesta proveniente dal *presenter* `guicontrol.AddressBookPanelPresenter` e sono successivamente impiegati al fine di popolare l'interfaccia grafica con i dati corrispondenti alla rubrica utente.

Componenti che ne fanno uso:

- CP02 – Rappresentazione dati
- CP03 – Gestione GUI

- `JSGroup`

Descrizione:

Classe avente il compito di rappresentare i gruppi della rubrica lato *presenter*, generata serializzando esclusivamente la parte che si vuole trasmettere ai client delle istanze del *transfer object* descritto sopra (`org.softwaresynthesis.mytalk.server.abook.Group`).

In particolare di un gruppo è possibile ottenere il nome e l'elenco dei contatti appartenenti al gruppo.

Gli oggetti `JSGroup` vengono creati a seguito di una richiesta proveniente dal *presenter* `guicontrol.GroupPanelPresenter` e sono successivamente impiegati al fine di popolare l'interfaccia grafica con la lista dei gruppi utente.

Componenti che ne fanno uso:

- CP02 – Rappresentazione dati
- CP03 – Gestione GUI

7.2.3 Package `org.softwaresynthesis.mytalk.clientpresenter.guicontrol`

- `guicontrol.AccountSettingsPanelPresenter`

Descrizione:

Si tratta del *presenter* richiamato dall'elemento grafico corrispondente al pannello `org.softwaresynthesis.mytalk.clientview.AccountSettingsPanel` per la gestione dei dati relativi all'utente associato al client.

Per questo motivo, l'oggetto `AccountSettingsPanelPresenter` ha il compito di mettere a disposizione per la vista le funzioni di *callback* che comportano la modifica di questi stessi dati.

Componenti che ne fanno uso:

- CP03 – Gestione GUI
- CV01 – GUI

- `guicontrol.AddressBookPanelPresenter`

Descrizione:

Tale *presenter* viene richiamato dall'elemento grafico corrispondente al pannello della rubrica (`org.softwaresynthesis.mytalk.clientview.AddressBookPanel`) e contiene la risposta a eventi quali la selezione di un contatto dalla rubrica da parte dell'utente. Contiene inoltre un'operazione di *refresh* che visualizza una lista di contatti ricevuta in input in forma grafica sulla UI.

Il *presenter* in questione rende inoltre possibile avviare una ricerca fra gli utenti registrati nel sistema nonché rilevare che l'operazione di ricerca è conclusa (e conseguentemente visualizzare nuovamente la rubrica).

I dati corrispondenti alla rubrica, ovvero le corrispondenti istanze di classi definite in Rappresentazione dati, se non sono presenti sul client devono essere recuperati dal server ed è responsabilità di questa classe contattare il Façade del Server al fine di ottenere le informazioni necessarie.

Infine in modo simile sono comunicati al server gli esiti delle operazioni di amministrazione sulla rubrica che hanno un effetto sulle informazioni di modello.

Componenti che ne fanno uso:

- CP03 – Gestione GUI
- CV01 – GUI

- `guicontrol.CallHistoryPanelPresenter`

  Descrizione:

  Questo *presenter* gestisce l'elemento corrispondente al pannello dello storico delle chiamate presente nell'interfaccia grafica, permettendo la visualizzazione di un *array* di chiamate corrispondenti alle comunicazioni di un determinato utente.

  Poiché il componente grafico supervisionato è totalmente passivo, non sono presenti operazioni di *callback* da associare alle componenti grafiche vere e proprie.

  Qualora i dati corrispondenti allo storico delle chiamate non fossero disponibili sul client che ospita questo *presenter* è responsabilità di quest'ultimo interrogare il server (attraverso il componente Façade del server) al fine di ottenere le informazioni necessarie.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – GUI

- `guicontrol.CommunicationPanelPresenter`

  Descrizione:

  Tale *presenter* si occupa della gestione del componente grafico corrispondente al pannello di comunicazione fra utenti. In particolare, comprende le operazioni necessarie all'aggiornamento della grafica corrispondente alle chat testuali e alle condivisioni, nonché alla visualizzazione dello *stream* video nel corso di una comunicazione audio/video o dell'immagine associata al profilo utente nel caso di una sola comunicazione audio.

  Le principali funzioni di *callback* utilizzate dall'interfaccia grafica presenti in questa classe corrispondono alla possibilità di interrompere una chiamata in corso, avviare/interrompere una condivisone di risorse, avviare/interrompere una registrazione, visualizzare/nascondere le aree di chat (che sono indipendenti dalla comunicazione corrente).

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – GUI

- `guicontrol.ContactPanelPresenter`

  Descrizione:

  Tale *presenter* ha la responsabilità di controllare l'aggiornamento della sezione dell'interfaccia grafica incaricata di visualizzare il profilo di un utente in risposta alla selezione dalla rubrica o dall'esito di una ricerca.

  Poiché inoltre, a partire da tale componente grafico è possibile avviare una comunicazione, tale classe comprende al proprio interno anche le funzioni che devono essere richiamate in risposta all'azione da parte dell'utente di dare avvio a una comunicazione.

  Poiché da `org.softwaresynthesis.mytalk.clientview.ContactPanel` è possibile anche svolgere alcune attività di amministrazione della rubrica utente, in questo *presenter* sono presenti anche le funzioni di *callback* necessarie per l'aggiunta e la rimozione di un contatto particolare ad un gruppo, bloccare o sbloccare tale contatto, nonché per l'aggiunta dello stesso alla rubrica se il suo profilo è stato aperto a seguito di una ricerca fra gli utenti registrati nel sistema.

Dal momento che, infine, tali operazioni di amministrazione hanno un effetto sulle entità del modello dei dati che risiedono sul server, tale classe ha la responsabilità di notificare gli aggiornamenti sul server al fine di mantenere la consistenza delle informazioni.

Componenti che ne fanno uso:

- CP03 – Gestione GUI
- CV01 – GUI

- `guicontrol.GroupPanelPresenter`

  Descrizione:

  Tale *presenter* ha la responsabilità di controllare l'aggiornamento della sezione dell'interfaccia grafica incaricata all'amministrazione dei gruppi creati dall'utente in risposta alla modifica (intesa come aggiunta o rimozione) di tali gruppi tramite le funzioni di *callback* associate.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – GUI

- `guicontrol.LoginPanelPresenter`

  Descrizione:

  Tale *presenter* ha il compito di ricevere dall'interfaccia grafica i dati di autenticazione, associando un suo metodo all'evento `onSubmit` del *form* di *login*, e in seguito inoltrarli alla sotto-architettura server tramite una *servlet* del componente Façade del server.

  L'oggetto `LoginPanelPresenter` permette semplicemente di comunicare al server i dati di *login* inseriti dall'utente.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV02 – Login

- `guicontrol.MainPanelPresenter`

  Descrizione:

  Classe base per i *presenter* associati ai diversi componenti dell'interfaccia grafica che contiene i metodi necessari all'inizializzazione di questi ultimi nonché i controlli che ne attivano la comparsa o scomparsa in risposta alle preferenze di visualizzazione dell'utente.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – GUI

- `guicontrol.MessagePanelPresenter`

  Descrizione:

  Tale *presenter* contiene le operazioni necessarie a visualizzare l'elenco dei messaggi presenti nella segreteria telefonica di un determinato utente.

  Poiché a partire dalla vista associata è possibile avviare/arrestare la riproduzione di un messaggio, questo oggetto deve contenere opportune funzioni di *callback* per rispondere alle richieste dell'utente in tal senso. La cancellazione di un messaggio, corrispondente al requisito RUFF15.4.0, è anch'essa possibile attraverso un'operazione di questo *presenter*, così come il cambiamento di stato del messaggio una volta che viene ascoltato.

  È prevista inoltre la possibilità di aggiornamento del componente grafico, in quanto la ricezione di nuovi messaggi può avvenire anche se l'utente è in linea (ad esempio se questi è impegnato in un'altra conversazione).

Qualora i dati corrispondenti alla segreteria telefonica di un utente non fossero disponibili sul client, è responsabilità di questa classe interrogare il server al fine di ottenere le informazioni necessarie nonché istanziare le opportune classi di Rappresentazione dati.

Infine, per le operazioni che hanno effetto sulla rappresentazione del modello dei dati sul server, quali la cancellazione di un messaggio o il cambiamento di stato di un messaggio, devono essere notificati al server attraverso il componente Façade del server.

Componenti che ne fanno uso:

- CP03 – Gestione GUI
- CV01 – GUI

- `guicontrol.PresenterMediator`

  Descrizione:

  L'istanza di tale classe ha il compito di gestire le collaborazioni fra i diversi *presenter* associati alle viste dell'interfaccia grafica e smistare i messaggi che questi si devono scambiare.

  A tale scopo, è necessario che tutti i *presenter* si registrino su di essa in modo da poter ricevere le successive richieste tramite chiamata di metodo. Inoltre, questi ultimi sono in grado di notificare gli eventi di interesse al `PresenterMediator` che resta costantemente in ascolto su di essi.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CP04 – Gestione remota

- `guicontrol.RegisterPanelPresenter`

  Descrizione:

  Tale *presenter* ha il compito di ricevere dall'interfaccia grafica i dati con cui desidera registrarsi un utente, associando un suo metodo all'evento `onSubmit` del *form* di registrazione, e in seguito inoltrarli alla sotto-architettura server tramite una *servlet* del componente Façade del server.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – Login

- `guicontrol.SearchResultPanelPresenter`

  Descrizione:

  Tale *presenter* ha la responsabilità di controllare l'aggiornamento della sezione dell'interfaccia grafica incaricata della ricerca di un contatto tra quelli iscritti al sistema dopo la richiesta specifica dell'utente.

  Per questo motivo, l'oggetto `SearchResultPanelPresenter` ha il compito di mettere a disposizione per la vista le funzioni di *callback* che comportano la modifica della stringa di ricerca di uno specifico contatto nella lista.

  Componenti che ne fanno uso:

  - CP03 – Gestione GUI
  - CV01 – GUI

- `guicontrol.ToolsPanelPresenter`

  Descrizione:

  Tramite tale *presenter* è possibile inizializzare la barra laterale degli strumenti rappresentata da `org.softwaresynthesis.mytalk.clientview.ToolsPanel`. Gli aggiornamenti cui tale porzione della vista è soggetta consistono nella visualizzazio-

ne/scomparsa di uno o più strumenti su richiesta dell'utente e per ognuno di questi è prevista la presenza di un'operazione.

D'altra parte, gli eventi scatenati dall'interazione dell'utente con il pannello degli strumenti che comportano la modifica dell'elemento mostrato nella regione centrale al posto del pannello principale, sono raccolti da questo *presenter* e inoltrati al `PresenterMediator` per la gestione.

In tal modo è possibile far comparire o scomparire il pannello di gestione dei dati personali corrispondenti al proprio account, della segreteria, la gestione dei gruppi e lo storico delle chiamate.

Poiché inoltre da `org.softwaresynthesis.mytalk.clienview.ToolsPanel` è possibile impostare il proprio stato, il *presenter* ad esso associato contiene le funzioni che notificano l'avvenuto cambiamento di stato al componente Façade del server.

Componenti che ne fanno uso:

- CP03 – Gestione GUI
- CV01 – GUI

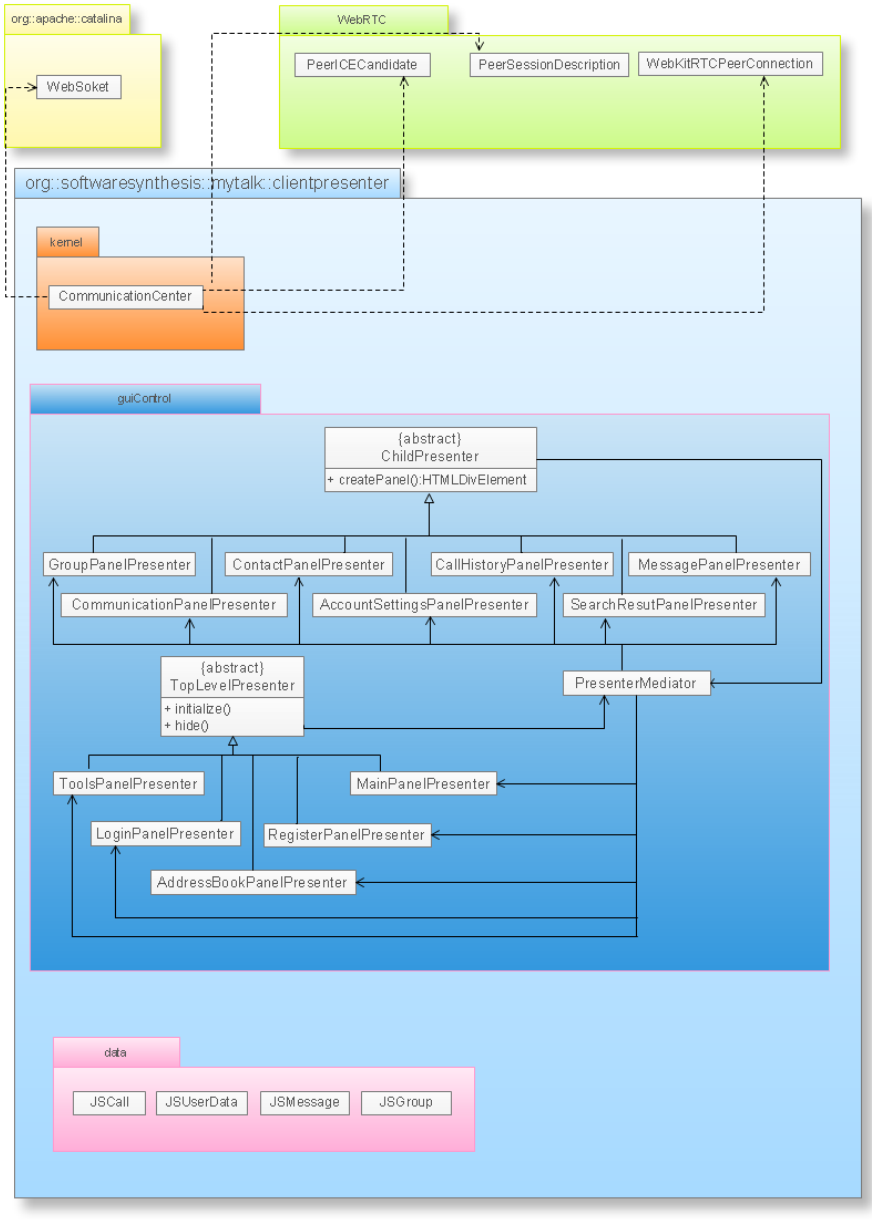

Figura 13: Diagramma delle classi - Architettura `mytalk.clientpresenter`

8 Architettura mytalk.clientview

Come accennato in precedenza, nella sezione 4 e 7, clientview ha il compito di definire la struttura di visualizzazione dei dati costituendo la GUI del sistema lato client.

Le motivazioni che hanno portato a separare tale sotto-architettura dalla parte logica sono da ricercare nei vantaggi di riutilizzo del codice e semplificazione della manutenzione:

- **future espansioni**: innanzitutto tale scelta permetterà ai progettisti di sviluppare (in futuro) molteplici tipologie di "viste", implementabili liberamente svincolando i programmatori dal conoscere al dettaglio la logica sottostante per la gestione delle comunicazioni.
- **semplificare la manutenzione**: così come scrivere da zero una nuova vista, anche modificare quelle già presenti risulta essere più facile, per gli stessi motivi descritti al punto precedente.

La sotto-architettura clientview rappresenta quindi una vista di *default* fornita dal team, con lo scopo di rappresentare in modo chiaro ed organizzato le possibilità di interazione da parte dell'utente finale con l'applicativo MyTalk.

Nell'utilizzare tale vista l'utente non avrà alcuna percezione di come sia stata progettata l'architettura totale del sistema, svincolandolo cosi dal dover conoscere le procedure necessarie all'esecuzione di una determinata operazione.

In figura 14 si riporta un abbozzo dell'interfaccia grafica utente realizzata mediante le classi di questa sotto-architettura. In conformità con quanto stabilito dai requisiti (RSDO10.0.0) l'interfaccia si sviluppa in un'unica pagina.

Figura 14: Rappresentazione ad alto livello della GUI

Per quanto riguarda la componentistica, la sotto-architettura clientview è costituita dai seguenti componenti:

- CV01 – GUI;
- CV02 – Login;

per le specifiche del quale si rimanda, rispettivamente, alle sezioni 8.1.1 e 8.1.2.

I nomi di tutte le classi riportate nella presente sezione sono inoltre implicitamente parte del package org.softwaresynthesis.mytalk.clientview, pertanto tale prefisso sarà omesso nella loro denominazione.

Analogamente a quanto osservato per la sotto-architettura clientpresenter, la rappresentazione tramite classi e oggetti è utilizzata a livello prettamente logico indipendentemente dal dominio applicativo concreto di implementazione.

In riferimento alla bozza di interfaccia utente presentata in figura 14 nella pagina precedente, la parte denominata `AddressBookPanel` conterrà la lista degli utenti nella rubrica, mentre `ToolsPanel` conterrà i componenti grafici che rappresentano tutte le funzionalità offerte dal sistema.

Il contenuto del `MainPanel`, invece, è destinato in relazione alla funzionalità scelta dall'utente in quel momento. Ad esempio, qualora l'utente selezioni un contatto presente nella rubrica, verrà visualizzato il profilo corrispondente al contatto scelto in `ContactPanel`, mentre nel caso in cui si attiva una forma di comunicazione, sarà visibile la parte grafica di un'istanza di `CommunicationPanel`.

8.1 Componenti evidenziati

8.1.1 CV01 – GUI

Descrizione:

Questo componente si occupa della presentazione dell'interfaccia grafica presentata all'utente finale. Le classi in esso contenute servono pertanto per la visualizzazione dei punti di accesso alle funzionalità offerte dal sistema all'utente.

Si noti che tale componente demanda tutta la logica di applicazione al *presenter* ed è quindi relativo solo ed esclusivamente alla grafica. Tutti i pannelli, che corrispondono ad altrettante viste, hanno pertanto un canale di comunicazione aperto con il *presenter* loro associato, e gli eventi generati dall'utente devono essere gestiti collegando l'evento alle opportune funzioni di *callback* offerte dai *presenter*.

Diagramma delle classi:

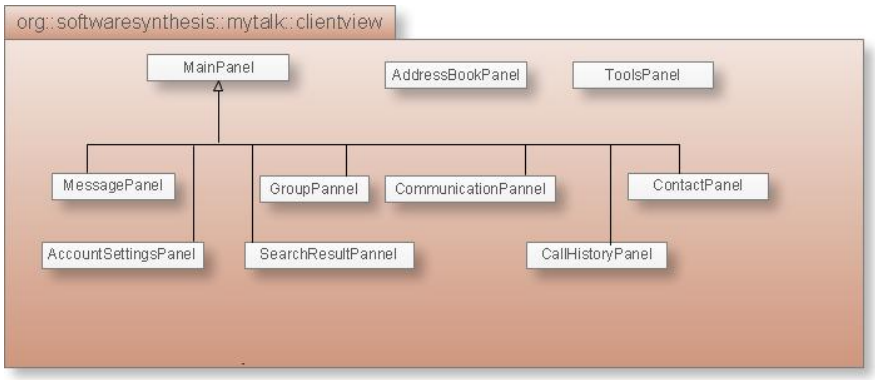

Figura 15: Diagramma delle classi - GUI

Classi utilizzate:

- `MainPanel`
- `ToolsPanel`
- `AddressBookPanel`
- `ContactPanel`
- `MessagePanel`
- `GroupPanel`
- `SearchPanel`
- `AccountSettingsPanel`
- `CallHistoryPanel`
- `CommunicationPanel`

8.1.2 CV02 – Login

Descrizione:

Il componente Login ha la responsabilità di ricevere in ingresso le credenziali di autenticazione immesse dall'utente del sistema e di trasmetterle al *presenter* ad esso associato `org.softwaresynthesis.mytalk.clientpresenter.guicontrol.LoginPanelPresenter`.

Diagramma delle classi:

Figura 16: Diagramma delle classi - Login

Classi utilizzate

- `LoginPanel`
- `RegisterPanel`

8.2 Descrizione delle classi

8.2.1 Package `org.softwaresynthesis.clientview`

- `AddressBookPanel`

Descrizione

Pannello che permette la visualizzazione della rubrica dell'utente connesso al sistema e di accedere ad alcune delle funzionalità di amministrazione della rubrica stessa (ad esempio l'aggiunta o la rimozione di gruppi).

In questo pannello sono anche presenti le funzionalità di ricerca fra gli utenti presenti nella rubrica, con la visualizzazione dei risultati della ricerca sostituita alla lista dei contatti presenti in rubrica.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `MainPanel`

Descrizione

Classe padre della gerarchia di oggetti grafici che possono comparire nella sezione principale dell'interfaccia utente. Ogni sua specializzazione definisce un nuovo tipo di pannello che costituisce l'elemento in primo piano dell'applicazione.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `ToolsPanel`

Descrizione

Classe che rappresenta il pannello degli strumenti della *home screen* dell'applicativo mediante il quale è possibile accedere alle funzionalità di ricerca, di segreteria telefonica, selezione della lingua, modifica dei dati dell'utente e storico delle chiamate.

All'occorrenza il pannello può essere messo in secondo piano e non essere più visibile per intero nella finestra principale. In questo pannello inoltre è visualizzato lo stato corrente dell'utente ed è possibile modificarlo per passare da "occupato" a "disponibile" o per disconnettersi e passare a *offline*.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `ContactPanel`

Descrizione

Sottoclasse di `MainPanel` utilizzata per rappresentare il profilo di un utente. Solo accedendo quest'ultimo sarà possibile avviare una comunicazione con l'utente. Il contatto viene visualizzato selezionando l'utente dalla rubrica oppure tra i risultati di una ricerca.

Attraverso questo pannello è inoltre possibile effettuare alcune operazioni di amministrazione sulla rubrica: aggiungere o rimuovere un utente ad uno dei gruppi della propria rubrica, bloccare un contatto o, se il profilo visualizzato corrisponde a un utente che non appartiene alla rubrica personale, aggiungerlo ad essa.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `MessagePanel`

Descrizione

Questa sottoclasse di `MainPanel` permette di accedere all'elenco dei messaggi in segreteria di un determinato utente e ne permette la gestione. La visualizzazione si attiva quando viene premuto il relativo pulsante mostrato dall'istanza di `ToolsPanel`.

Tramite questo elemento grafico è possibile anche riprodurre il contenuto di un messaggio, dopo che quest'ultimo è stato scaricato dal server (se non era già presente in memoria), arrestarne la riproduzione oppure cancellarlo se non si desidera che venga conservato.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `GroupPanel`

Descrizione

Tramite questo pannello è possibile visionare i gruppi che compongono la rubrica dell'utente. I gruppi vengono rappresentanti mediante il loro nome all'interno di una lista e affianco ad ogni nome comparirà un icona atta ad indicare la possibilità di eliminare il gruppo selezionato.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `SearchPanel`

Descrizione

Tramite questo pannello grafico, dotato di un <input> per l'inserimento di una parola chiave, è possibile ricercare nella propria rubrica tutti i contatti contenenti nel nome, cognome o indirizzo mail, la parola da ricercare. Una ricerca tramite questo pannello aggiorna il MainPanel con la lista dei contatti corretti secondo la ricerca.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `AccountSettingsPanel`

Descrizione

Questa classe è utilizzata per rappresentare il *form* che si presenta all'utente per la modifica dei propri dati, costituisce inoltre una sottoclasse di `MainPanel` e viene visualizzata quando l'utente preme il relativo pulsante nel pannello degli strumenti.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `CallHistoryPanel`

Descrizione

Questa classe estende `MainPanel` e viene impiegata per rappresentare lo storico delle chiamate visualizzato quando l'utente attiva il pulsante corrispondente presente nel `ToolsPanel`. Oltre alla visualizzazione dello storico delle chiamate non è prevista alcuna forma di interazione attiva con l'utente.

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `CommunicationPanel`

Descrizione

Tale componente grafico si attiva nel momento in cui è stata avviata una forma di comunicazione, sia essa audio, audio/video oppure testuale. Per le comunicazioni audio e audio/video consente la visualizzazione dell'immagine associata al profilo dei destinatari oppure dei relativi *stream* video e permette di visualizzare le risorse condivise.

Inoltre tale entità grafica prevede anche la presenza di una o più aree di chat che non corrispondono necessariamente alla comunicazione in corso (perché è possibile avere più comunicazioni testuali in parallelo fra loro e con una comunicazione multimediale).

Componenti che ne fanno uso:

- CV01 – GUI
- CP03 – Gestione GUI

- `LoginPanel`

Descrizione

Per la descrizione di questo elemento grafico si rimanda alla sezione 8.1.2 a pagina 51.

Componenti che ne fanno uso

- CV02 – Login

- `RegisterPanel`

Descrizione

Pannello contenente la form per la registrazione utente.

Componenti che ne fanno uso

- CV02 – Login

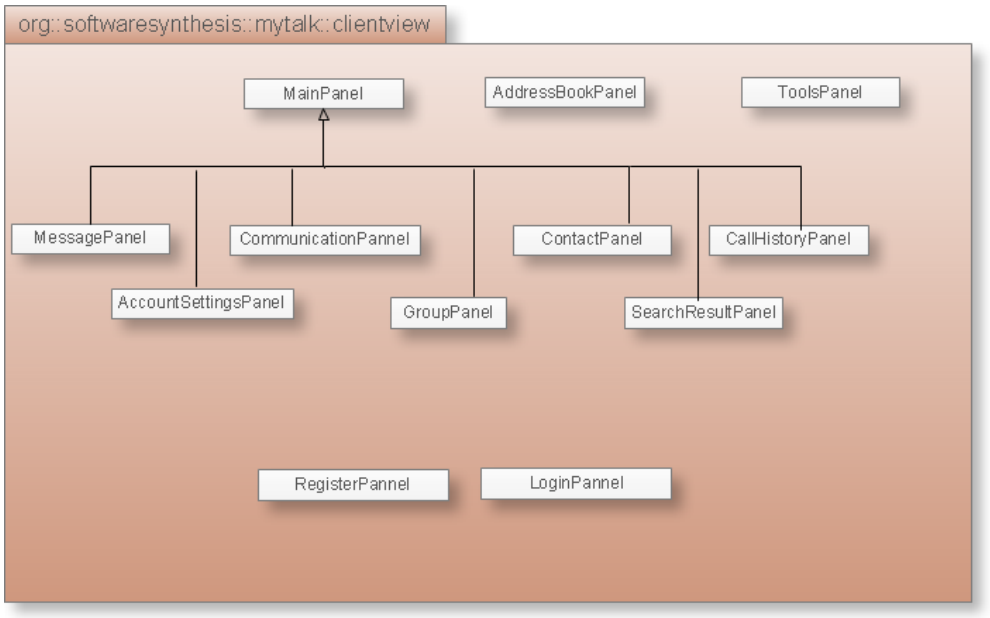

Figura 17: Diagramma delle classi - Architettura `mytalk.clientview`

9 Design pattern utilizzati

In conformità con quanto stabilito nella sezione 7.2 del documento *norme_di_progetto.3.0.pdf*, per ognuno dei *design pattern* sono riportati lo scopo e una descrizione del contesto di applicazione del *pattern* all'interno di un componente, evidenziando in particolar modo i vantaggi derivati e riportando un diagramma esemplificativo.

Per una visione d'insieme dei componenti utilizzati da un *pattern*, e dei *pattern* utilizzati da un componente, si rimanda alle sottosezioni "Tracciamenti Componenti-Design Pattern" e "Tracciamenti Design Pattern-Componenti" della sezione 11 a pagina 77.

9.1 Data Access Object (DAO)

9.1.1 Scopo

Il *pattern* DAO ha lo scopo di disaccoppiare la logica di *business* dalla logica di accesso ai dati, questo si ottiene spostando la logica di accesso ai dati dai componenti di *business* stessi ad una classe DAO ad hoc per il DBMS scelto, rendendo i componenti che implementano la *business logic* indipendenti dalla natura del dispositivo di persistenza.

Un simile approccio garantisce che un eventuale cambiamento del dispositivo di persistenza non comporti modifiche sui componenti di *business*.

9.1.2 Componenti che lo implementano

CS01 – Gestione database

Le classi DAO consentono di isolare l'accesso alle tabelle del database dalla parte di *business logic* corrispondente alle entità TO (*transfer object*), incapsulando in metodi di alto livello le operazioni sui record del database sottostante.

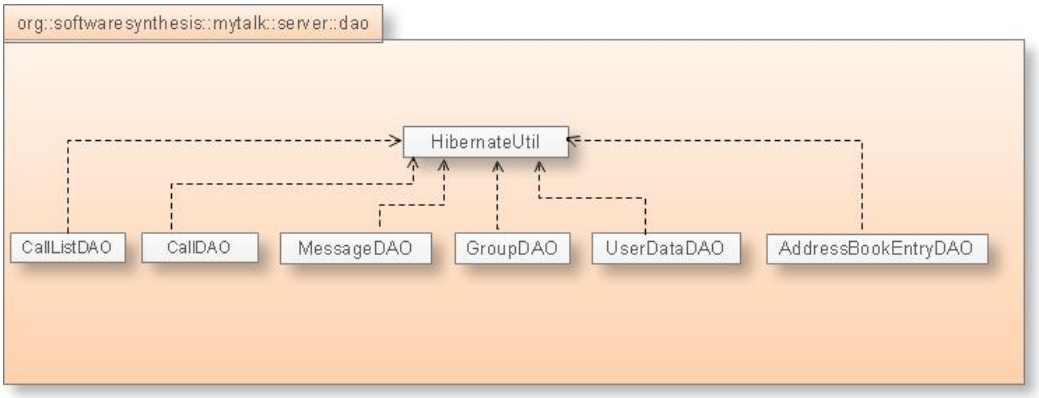

Figura 18: Applicazione del *pattern* Data Access Object

Operando una distinzione di questo tipo gli utenti esterni possono considerare gli oggetti che richiedono la mappatura nel database alla stregua di normali oggetti Java (POJO) senza preoccuparsi del meccanismo di persistenza dei dati.

9.2 Façade

9.2.1 Scopo

Fornire un'interfaccia unificata per un insieme di interfacce o classi presenti in una sotto-architettura, nascondendo i dettagli implementativi di quest'ultima.

Façade definisce inoltre un'interfaccia di livello più alto che rende la sotto-architettura più semplice da utilizzare e riduce il numero di dipendenze funzionali fra i componenti appartenenti a sotto-architetture esterne e le classi del server.

9.2.2 Componenti che lo implementano

CS07– Façade del server

Questa componente della sotto-architettura server fornisce una serie di punti di accesso centralizzati (*servlet*) per i componenti Gestione rubrica, Gestione connessione, Gestione segreteria, Gestione chiamate e Gestione autentcazione secondo la tabella riportata nella sezione 6.1.7 a pagina 22. In pratica è possibile pensare ad ogni *servlet* come ad un façade atto a nascondere l'iter procedurale per la realizzazione di una determinata operazione.

Classi del package `org.softwaresynthesis.mytalk.server.abook.servlet`

Ogni classe di tale package implementa Façade poiché nasconde l'iter per la gestione della rubrica svincolando l'utilizzatore dal doverlo conoscere. Nello specifico tali classi richiamano in più punti e in maniera sequenziale, classi presenti nel package `server.abook` e `server.dao`.

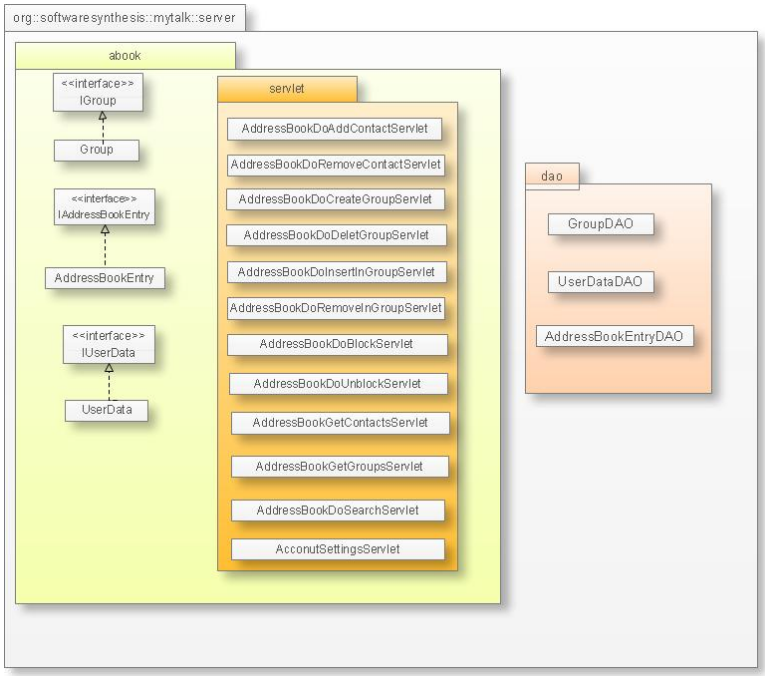

Figura 19: Applicazione del *pattern* Façade a Gestione rubrica

Essendo già il diagramma sufficientemente complesso si ritiene che l'aggiunta di tutte le frecce che rappresentano le dipendenza lo rendano completamente illeggibile, per tale ragione le dipendenze si riportano in forma di elenco, in particolare:

- la *servlet* `AddressBookDoAddContactsServlet` ha una dipendenza da `AddressBookEntry`, `IGroup`, `IUserData`, `GroupDAO` e `UserDataDAO`;
- `AddressBookDoBlockServlet` dipende da `IAddressBookEntry`, `IUserData` e `UserDataDAO`;
- `AddressBookDoCreateGroupServlet` dipende da `Group`, `IGroup`, `IUserData` e `GroupDAO`;
- la *servlet* `AddressBookDoDeleteGroupServlet` dipende invece da `IAddressBookEntry`, `IGroup`, `AddressBookEntryDAO` e `GroupDAO`;

- `AddressBookDoInsertInGroupServlet` dipende da `AddressBookEntry`, `IAddressBookEntry`, `IGroup`, `IUserData`, `GroupDAO` `UserDataDAO`;
- la *servlet* `AddressBookDoRemoveContactServlet` dipende da `IAddressBookEntry`, `IUserData`, `AddressBookEntryDAO` e `UserDataDAO`;
- `AddressBookDoRemoveFromGroupServlet` dipende da `AddressBookEntry`, `IAddressBookEntry`, `IGroup`, `IUserData`, `GroupDAO` e `UserDataDAO`;
- `AddressBookDoSearchServlet` dipende da `IUserDataDAO` e `UserDataDAO`;
- `AddressBookDoUnblockServlet` dipende da `IAddressBookEntry`, `IUserData` e `UserDataDAO`;
- `AddressBookGetContactsServlet` dipende da `IAddressBookEntry` e `IUserData`;
- `AddressBookGetGroupsServlet` dipende da `IAddressBookEntry`, `IGroup`, `IUserData` e `GroupDAO`;
- `AccountSettingsServlet` dipende da `IUserData` e `UserDataDAO`.

Classi del package `org.softwaresynthesis.mytalk.server.message.servlet`

Ogni classe di tale package implementa Façade poiché nasconde l'iter per la gestione della segreteria telefonica svincolando l'utilizzatore dal doverlo conoscere, nello specifico tali classi richiamano in più punti e in maniera sequenziale classi presenti nel package `server.message` e `server.dao`.

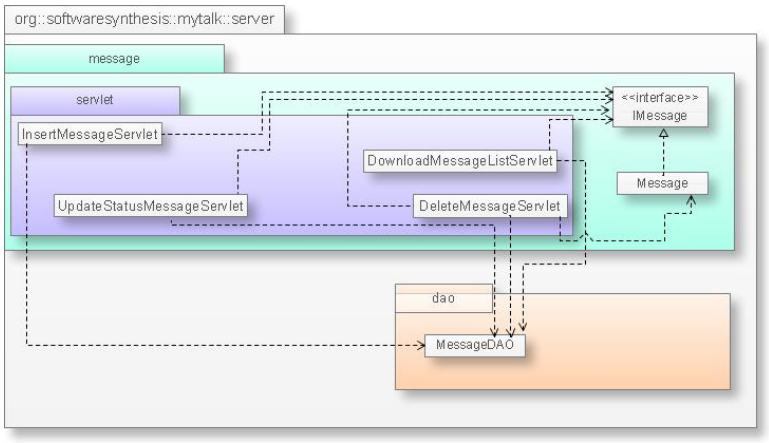

Figura 20: Applicazione del *pattern* Façade a Gestione segreteria

Classe `org.softwaresynthesis.mytalk.server.connection.ChannelServlet`

Tale classe nasconde la complessità di gestione di una comunicazione client-server, nello specifico svincola l'utilizzatore dal dover conoscere l'iter di utilizzo delle classi di `server.abook` e `server.connection`.

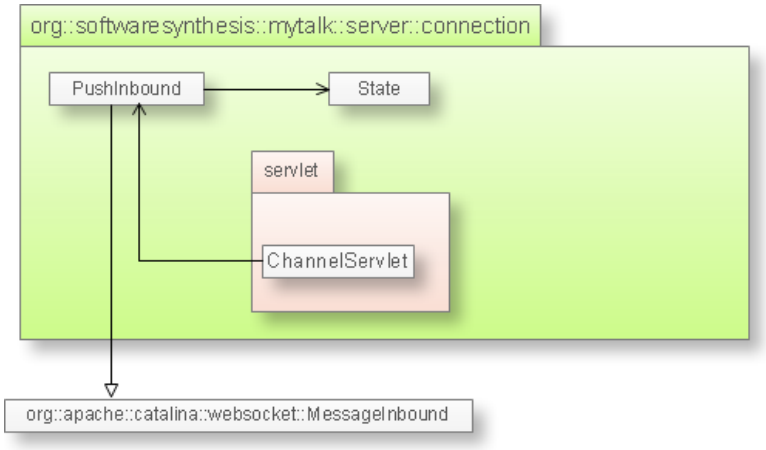

Figura 21: Applicazione del *pattern* Façade a Gestione connessione

Classe `org.softwaresynthesis.mytalk.server.call.servlet.DownloadCallHistoryServlet`

Tale classe nasconde la complessità di gestione di una richiesta di *download* della lista delle chiamate. Nello specifico svincola l'utilizzatore dal dover conoscere l'iter di utilizzo delle classi di `server.abook`, `server.dao` e `server.call`.

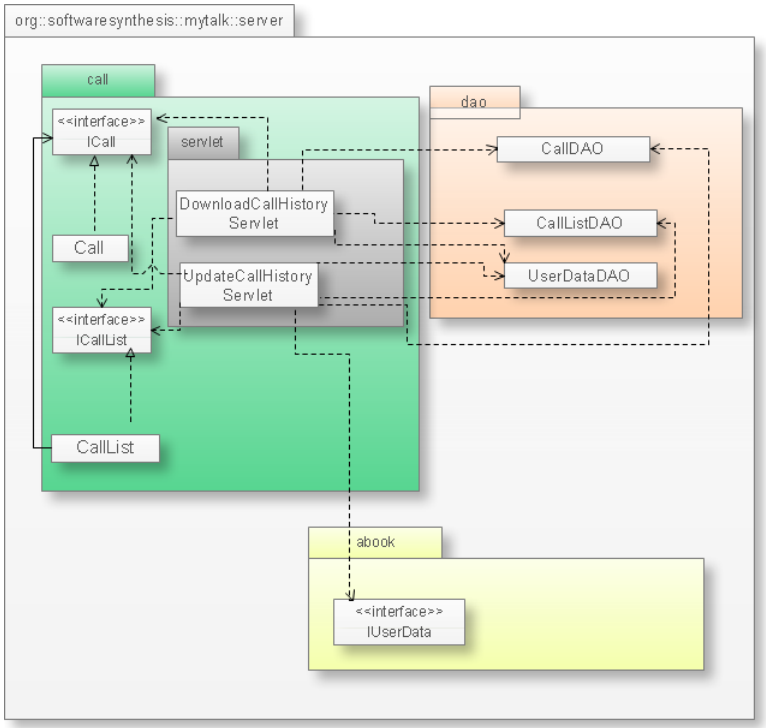

Figura 22: Applicazione del *pattern* Façade a Gestione chiamate

Classi del package `org.softwaresynthesis.mytalk.server.authentication.servlet`

Ogni classe di tale package implementa Façade poiché nasconde l'iter per l'autenticazione (*login*, *logout* e registrazione) di un utente svincolando l'utilizzatore dal doverlo conoscere. Nello

specifico tali classi richiamano in più punti e in maniera sequenziale, classi presenti nel package `server.abook`, `server.dao` e `server.autenthication`.

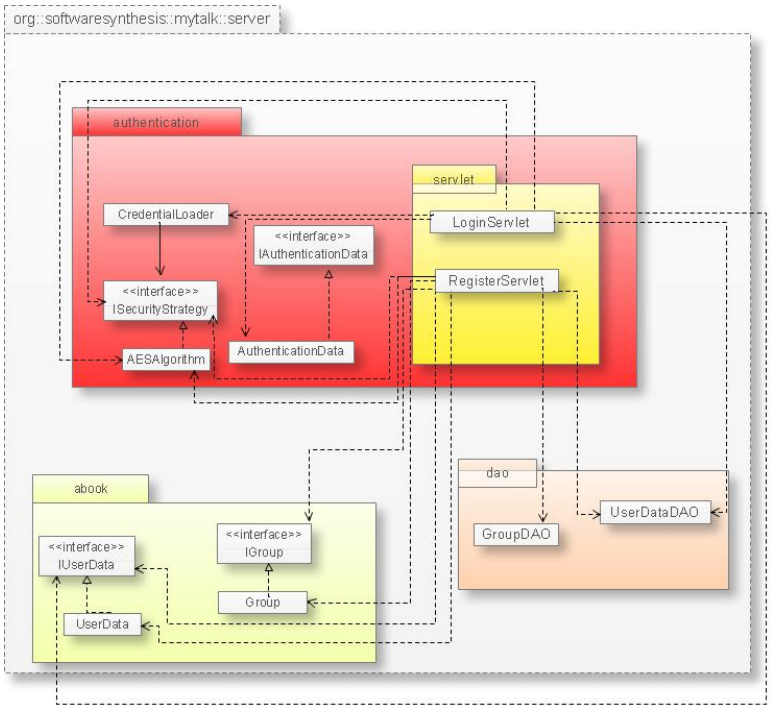

Figura 23: Applicazione del *pattern* Façade a Gestione autenticazione

9.3 Factory Method

9.3.1 Scopo

Definisce un'interfaccia per la creazione di un oggetto, lasciando alle sottoclassi la decisione sulla classe concreta che deve essere istanziata e consente di deferire l'istanziazione di una classe alle sottoclassi.

9.3.2 Componenti che lo implementano

CS02 – Gestione connessione
Gli oggetti che intendono rappresentare connessioni sono sottotipi di della classe di libreria `org.apache.catalina.websocket.StreamInbound` e vengono instanziati mediante un Factory Method nelle sottoclassi di `org.apache.catalina.websocket.WebSocketServlet`.

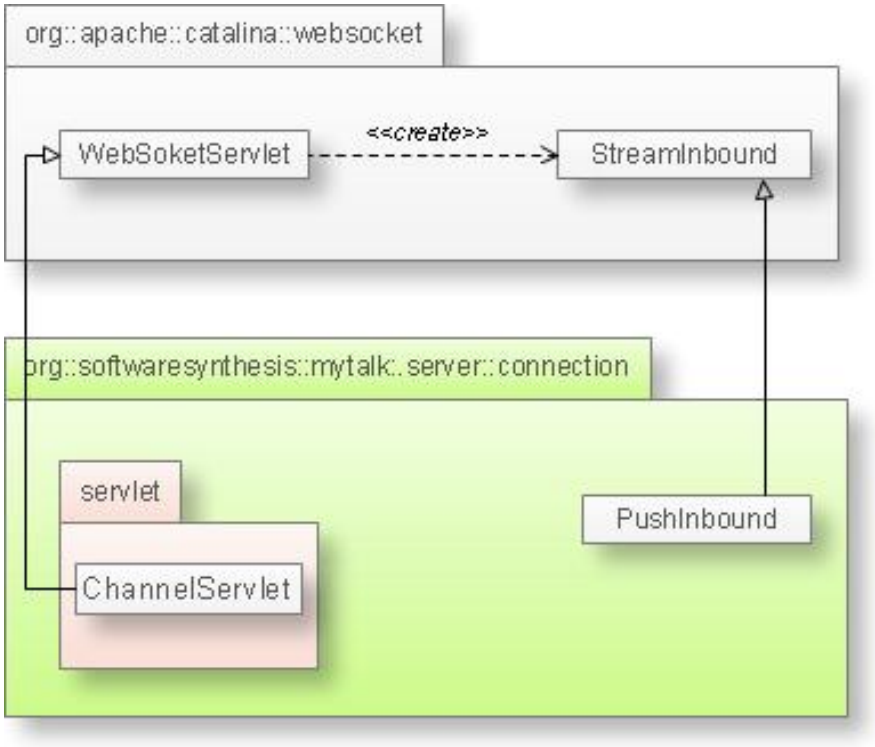

Figura 24: Applicazione del *pattern* Factory Method

In particolare, la classe `connection.ConnectionManager` restituisce un oggetto che rappresenta la connessione di tipo dinamico `connection.PushInbound` che implementa il comportamento come reazione ai messaggi in ingresso facendo a sua volta *overriding* del metodo `onTextMessage(CharBuffer)`.

9.4 Strategy

9.4.1 Scopo

Definisce un'interfaccia per il comportamento di un algoritmo generico, lasciando alle classi implementanti il compito di definirne la strategia d'implementazione.

9.4.2 Componenti che lo implementano

CS04 – Gestione autenticazione
All'interno di tale classe l'utilizzo del pattern si ha nel momento in cui è necessario definire una strategia di criptazione e decriptazione dei dati, passati dall'utente per eseguire l'autenticazione. L'interfaccia atta a definire l'algoritmo generico è `authentication.ISecurityStrategy` mentre una prima implementazione proposta dal team è `authentication.AESAlgorithm`, che definisce un algoritmo di criptazione basato su AES.

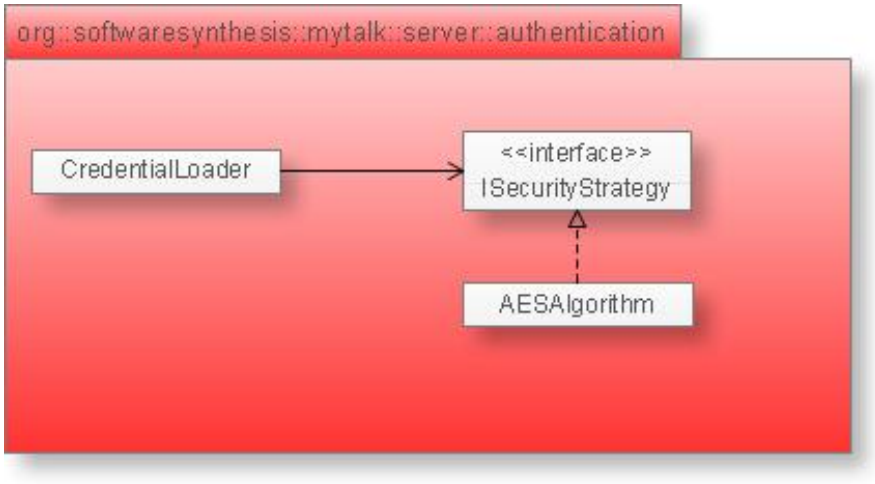

Figura 25: Applicazione del *pattern* Strategy

9.5 Model-View-Presenter

9.5.1 Scopo

Il pattern architetturale *Model-View-Presenter* similmente a quanto accade per *Model-View-Controller* (MVC), ha lo scopo di mantenere separata la *business logic*, cioè la gestione dei dati secondo le regole di un determinato dominio e la loro memorizzazione in forma persistente, dalla presentazione e manipolazione mediante interfaccia utente.

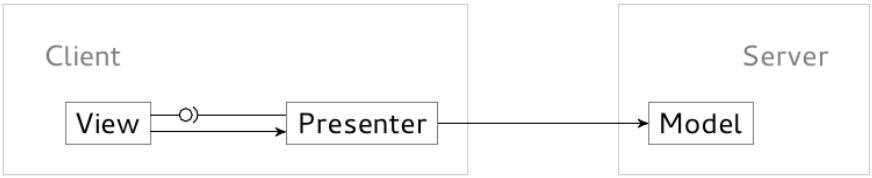

Figura 26: Diagramma ad alto livello del *pattern* MVP

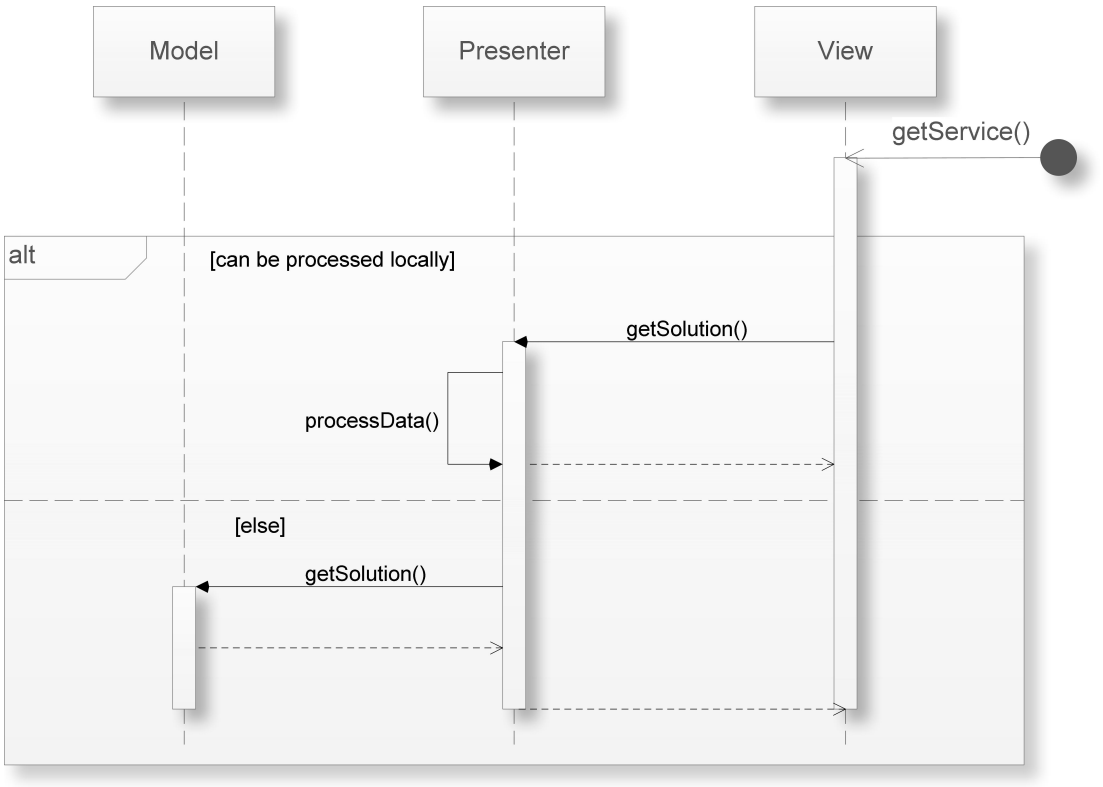

Figura 27: Diagramma di sequenza che illustra le collaborazioni in MVP

Come si evince dal diagramma riportato in figura 27 le interazioni avvengono solo tra *view* e *presenter* oppure tra *presenter* e *model*, senza che avvenga mai uno scambio di dati diretto fra *model* e *view*.

Ciò si deve al fatto che la *business logic* e il modello dei dati risiedono nel server mentre il *presenter* e la *view* sono situati nel client. Quando un utente richiede un servizio tramite l'interfaccia grafica, la richiesta viene inoltrata dalla *view* al *presenter*.

Qualora quest'ultimo fosse in grado di soddisfare tale richiesta con le risorse di cui dispone, i dati sono restituiti immediatamente alla *view* senza alcun bisogno di richiedere l'intervento del server. Nel caso, invece, in cui il *presenter* non fosse in grado di servire autonomamente la *view*, interrogherebbe il server al fine di ottenere i dati da restituire alla componente grafica.

Il vantaggio di un simile schema di interazione consiste nella riduzione del traffico di rete e nel conseguente incremento delle prestazioni in termini di velocità e, di conseguenza, dell'esperienza utente in generale.

9.5.2 Componenti che lo implementano

MVP viene utilizzato come il *pattern* più ad alto livello del sistema: la distinzione fra *model*, *presenter* e *view* è infatti rispecchiata dalla suddivisione del sistema nelle tre sotto-architetture `server`, `clientpresenter` e `clientview`.

In generale, l'utilizzo di MVP riduce l'accoppiamento tra le sotto-architetture minimizzando le modifiche richieste a ognuno di essi come conseguenza di cambiamenti all'interno degli altri.

Inoltre i componenti di questa sotto-architettura non sono vincolati a utilizzare la rete per accedere alle informazioni che sono memorizzate sul server quando queste sono già disponibili (e possono essere elaborate) sul client, migliorando quindi l'esperienza utente.

Le componenti del sistema che prendono parte alle collaborazioni previste dal *pattern* MVP sono

CS01 – Gestione database
Componente che ha il ruolo di gestire la persistenza dei dati sul server necessaria alla memorizzazione delle entità di interesse del modello dei dati.

CS06 – Gestione chiamate, CS03 – Gestione rubrica e CS05 – Gestione segreteria
Contengono le rappresentazioni delle entità della *business logic* lato server che devono essere interrogate per ricavare le informazioni necessarie ai client in forma opportuna.

CS07 – Façade del server
Componente del server interessato dalla ricezione delle richieste da parte dei *presenter*, recupera le informazioni di cui questi ultimi necessitano e le restituisce in forma serializzata e compatibile con il dominio applicativo dei client.

CP04 – Gestione comunicazione
Intercetta le comunicazioni entranti verso il client a partire dal server che non hanno origine in una richiesta esplicita da parte del client come, ad esempio, gli aggiornamenti di stato dei diversi utenti o le chiamate in ingresso.

CP03 – Gestione GUI
Riceve i comandi dalla vista ed è responsabile del suo aggiornamento e dell'interrogazione del server nel momento in cui i dati necessari per popolare l'interfaccia utente non siano disponibili sul client. Questo componente integra al suo interno gli oggetti *presenter* specifici per ogni vista e ne gestisce la collaborazione.

GUI
Corrisponde all'insieme delle viste ed è accessibile al *presenter* tramite le DOM API di JavaScript.

9.6 Singleton

9.6.1 Scopo

Il *pattern* creazionale Singleton, garantisce che una determinata classe possa essere istanziata una sola volta, e di fornirne un punto di accesso globale. Questo *pattern* va utilizzato negli ambiti in cui si ha la necessità che l'accesso ad una determinata entità sia unico, in modo da permettere la gestione ottimale della risorsa stessa.

9.6.2 Componenti che lo implementano

CS01 – Gestione database
La classe `server.dao.HibernateUtil` è implementata come Singleton dal momento che si desidera che in ogni momento ne sia attiva un'unica istanza. Tramite i metodi che essa mette a disposizione, le classi DAO sono in grado di procurarsi un riferimento alla sessione di connessione al DBMS al fine di effettuare le transazioni di cui necessitano.

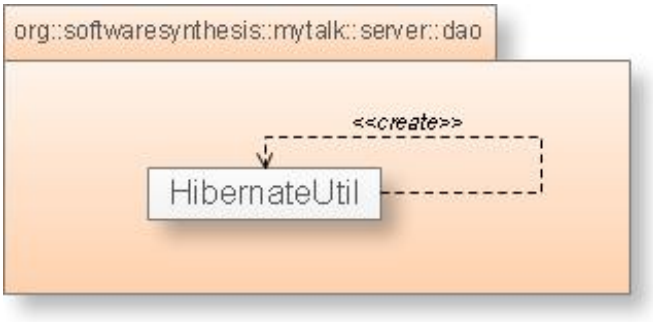

Figura 28: Applicazione del *pattern* Singleton a Gestione database

10 Diagrammi delle attività

In questa sezione saranno descritti i diagrammi di attività che rappresentano il flusso di utilizzo dei vari servizi messi a disposizione dal prodotto MyTalk.

10.1 Diagramma di attività generale

Il diagramma in figura 29 rappresenta il flusso principale dell'applicazione MyTalk. L'accesso alle funzionalità del sistema è vincolato allo svolgimento con successo dell'azione di autenticazione, illustrata con maggiori dettagli nel diagramma di sotto-attività 30 nella pagina successiva.

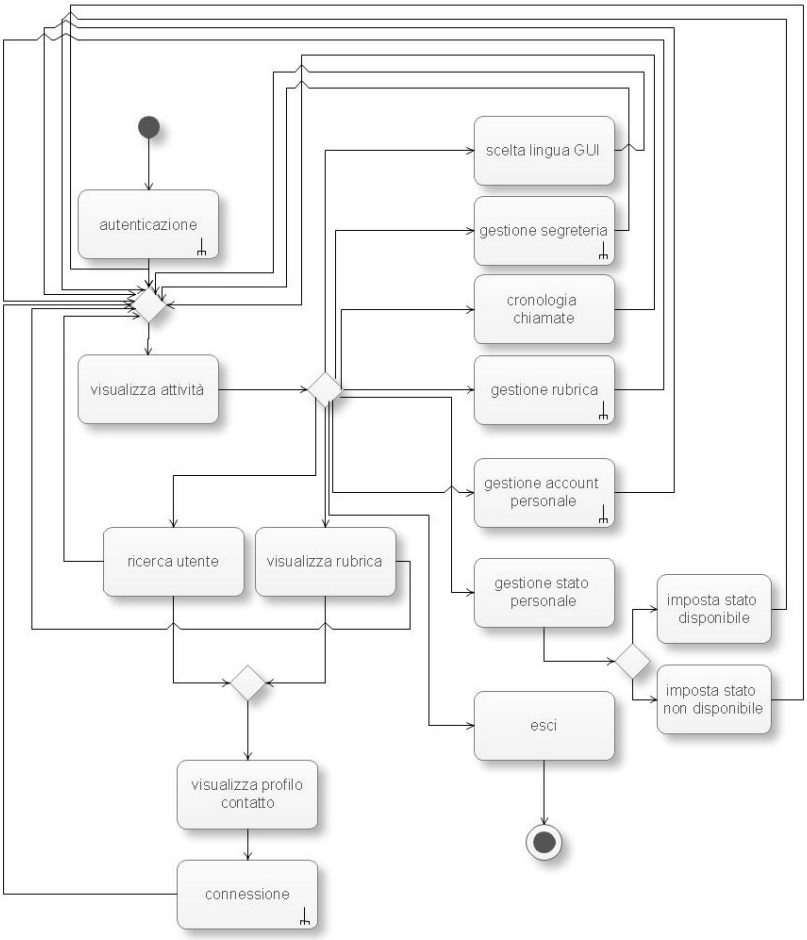

Figura 29: Diagramma di attività generale che descrive l'interazione con il sistema

A seguito dell'autenticazione, diviene possibile scegliere una fra le azioni di amministrazione, vale a dire

- scelta lingua GUI;
- gestione segreteria (illustrata dal diagramma 35);
- gestione rubrica (illustrata dal diagramma 33);
- gestione account (illustrata dal diagramma 34);
- gestione dello stato personale;

oppure di consultazione informazioni, in particolare:

- visualizzazione dello storico delle chiamate;
- visualizzazione della rubrica;
- ricerca di un utente.

A partire infine dalla visualizzazione del profilo di un utente, cui è possibile accedere tanto tramite una ricerca quanto dall'elenco dei contatti in rubrica, è possibile dare inizio a una comunicazione (diagramma 36 a pagina 71).

10.2 Diagrammi di attività Autenticazione

Il diagramma riportato in figura 30 illustra la sotto-attività di autenticazione di un utente al sistema. A partire dalla schermata iniziale è possibile inserire le proprie credenziali di accesso al sistema se ne si è provvisti, richiederle se è la prima volta che si effettua l'accesso al sistema (avviando la procedura di registrazione riportata in figura 31) oppure recuperarle (sotto-attività in figura 32 a pagina 68).

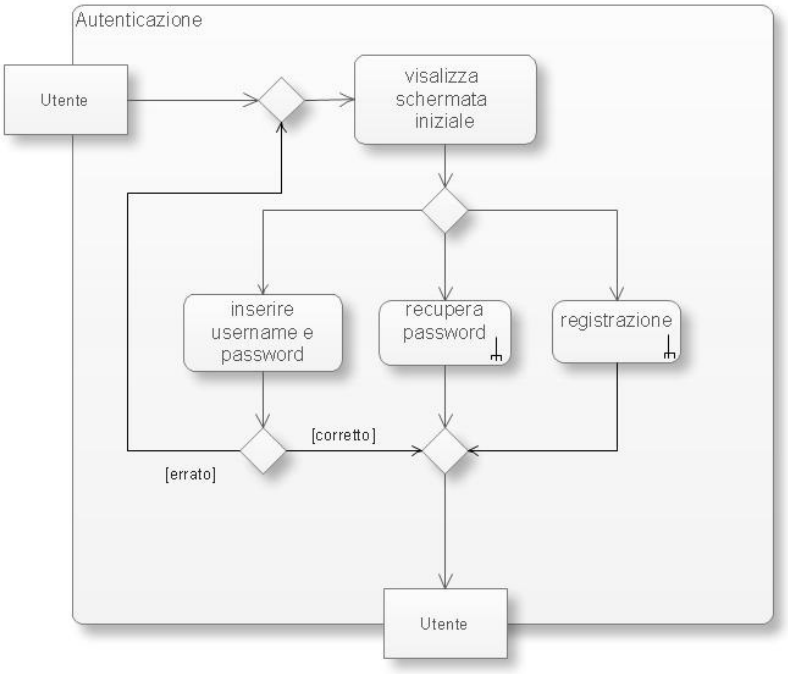

Figura 30: Diagramma di attività relativo all'autenteticazione

10.2.1 Diagramma di attività Registrazione

In dettaglio, la registrazione di un utente al sistema (fig. 31 nella pagina successiva) prevede l'inserimento di una serie di dati alcuni dei quali obbligatori (indirizzo email, password, domanda segreta e relativa risposta) e altri facoltativi (nome, cognome e immagine del profilo). Al termine della procedura di registrazione l'utente dispone di un account personale e delle credenziali di accesso allo stesso.

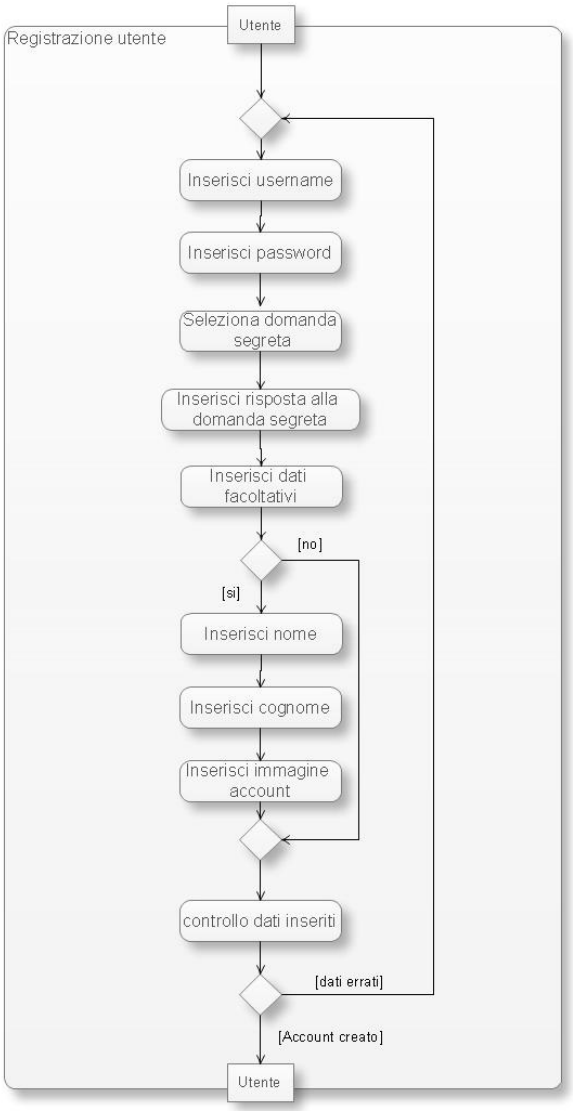

Figura 31: Diagramma di attività relativo alla registrazione

10.2.2 Diagramma di attività Recupero password

Il recupero della password (fig. 32 nella pagina seguente) prevede invece l'inserimento dell'email, la visualizzazione della domanda segreta impostata in fase di creazione dell'account e l'inserimento della relativa risposta. In caso di risposta corretta si ha il termine della sotto-attività con l'invio dei dati richiesti via email, in caso contrario la procedura deve essere ripetuta.

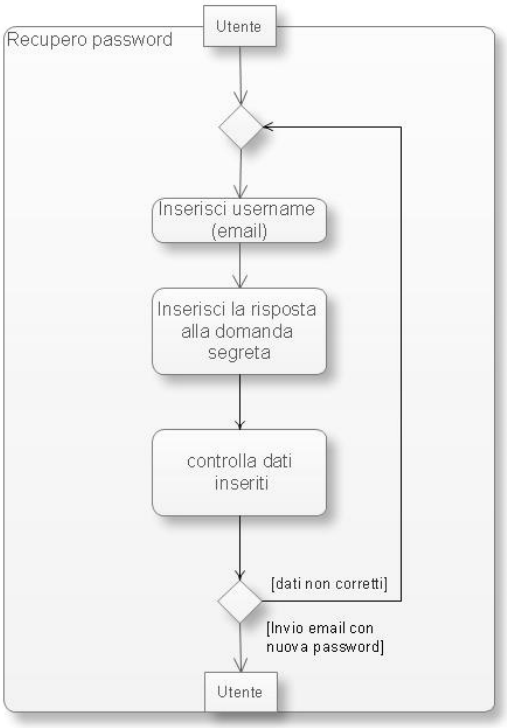

Figura 32: Diagramma di attività relativo al recupero della password

10.3 Diagramma di attività Gestione rubrica

La sotto-attività di gestione (diagramma in fig. 33 nella pagina successiva) della rubrica prevede l'accesso a due classi di azioni, a seconda che sia o meno richiesta da parte dell'utente una conferma esplicita per il completamento dell'attività.

In particolare, la prima categoria di azioni comprende:

- l'aggiunta di un contatto alla rubrica;
- l'ordinamento della rubrica;
- l'eliminazione di un contatto;
- la ricerca di un contatto;
- l'inserimento di un contatto in un gruppo;

mentre la seconda categoria raggruppa azioni quali:

- l'eliminazione di un gruppo;
- la modifica di un gruppo;
- l'importazione della rubrica da un file locale in formato XML;
- l'esportazione della rubrica personale in un file XML.

La risposta affermativa alla domanda di conferma in seguito alla scelta di un'azione della seconda categoria porta alla conclusione della sotto-attività, mentre una risposta negativa comporta la scelta di una nuova azione amministrativa.

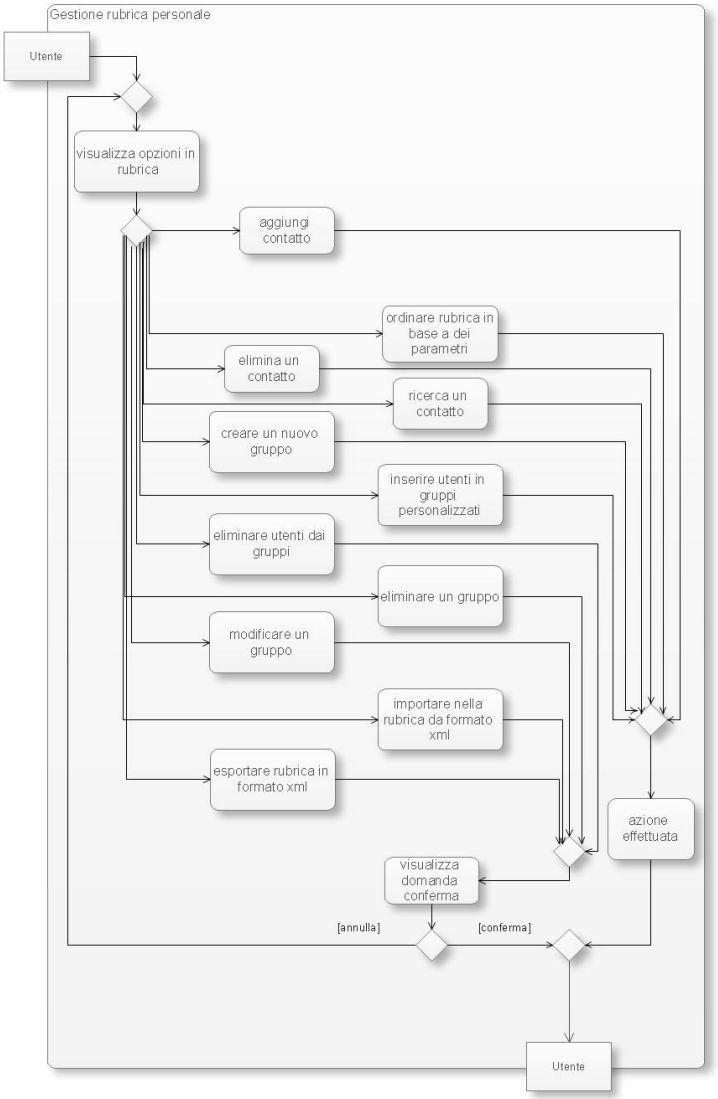

Figura 33: Diagramma di attività relativo alla gestione della rubrica personale

10.4 Diagramma di attività Gestione account personale

La gestione dell'account personale (fig. 34 nella pagina seguente) comporta in primo luogo la visualizzazione di una schermata che mette a disposizione le operazioni disponibili, quindi la scelta di una di queste ultime. Le azioni che possono essere effettuate sono, in particolare:

- modifica della password utente;
- modifica dei dati anagrafici;
- modifica dell'immagine del profilo;
- modifica della domanda segreta e/o della relativa risposta per il recupero della password.

Al termine dell'operazione è prevista la visualizzazione di una domanda di conferma e, in caso di risposta affermativa da parte dell'utente, la modifica va a buon fine. In caso contrario, l'operazione non sortisce alcun effetto e si ritorna alla possibilità di scegliere l'azione amministrativa da effettuare.

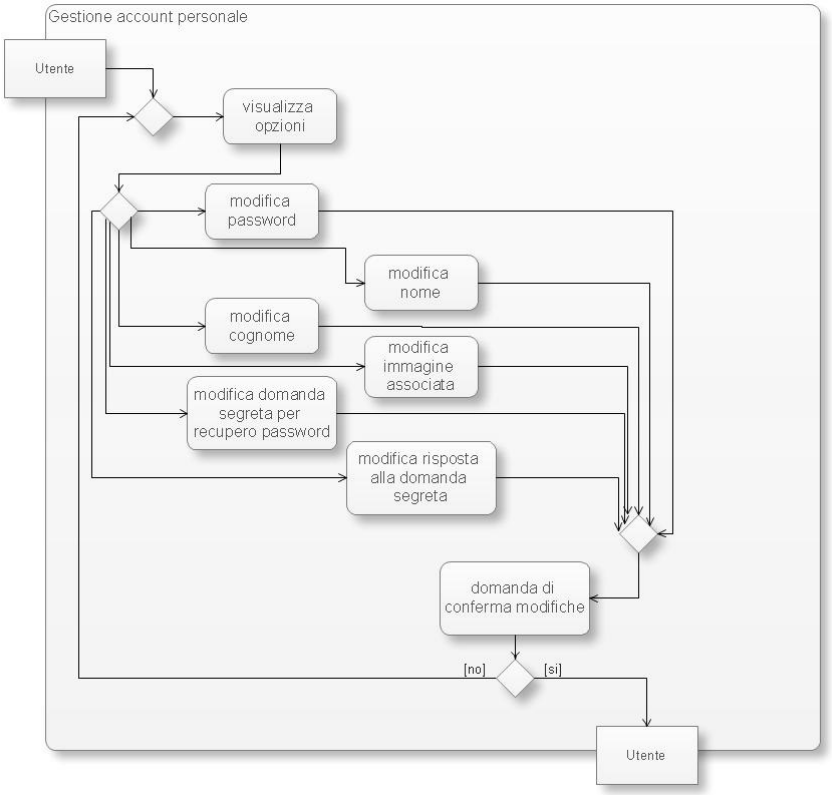

Figura 34: Diagramma di attività relativo alla gestione dei dati dell'account personale

10.5 Diagramma di attività Gestione segreteria

La gestione della segreteria prevede la scelta fra le seguenti azioni:

- ascoltare un messaggio della segreteria;
- cancellare un messaggio;
- impostare lo stato (ascoltato/non ascoltato) di un messaggio;

Compiere una di queste azioni comporta l'uscita dalla sotto-attività ma l'esecuzione di più di una di queste azioni in sequenza è comunque possibile grazie al *merge* antecedente l'azione "Visualizza attività" nel diagramma di attività generale riportato in figura 29 a pagina 65.

Figura 35: Diagramma di attività relativo alla gestione della segretria

10.6 Diagrammi di attività Connessione

L'utente ha inoltre la possibilità di connettersi con altri utenti tramite l'attività di connessione.

Come si evince dal diagramma riportato in figura 36, la comunicazione con un utente dipende intrinsecamente dallo stato in cui si trova: se questo è *offline* oppure *online* ma occupato, può essere raggiunto in maniera indiretta solo tramite la segreteria telefonica (diagramma 43), mentre se l'utente è disponibile può essere contattato immediatamente.

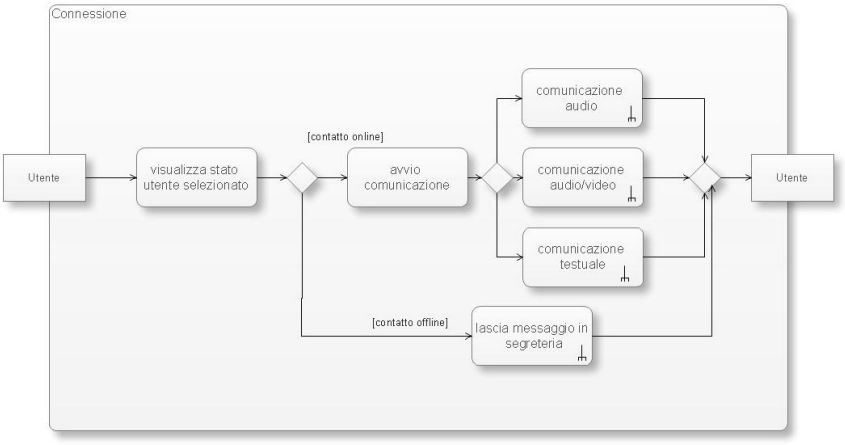

Figura 36: Diagramma di attività relativo alla connessione

Esistono tre tipologie di comunicazione a seconda del mezzo utilizzato:
- audio (illustrata nel diagramma in fig. 37);
- audio/video (diagramma in fig. 38);
- testuale (diagramma in fig. 39);

Tutte e tre le tipologie di connessione prevedono la condivisione di risorse secondo le modalità riportate in figura 40. Inoltre, nel corso di una comunicazione di tipo audio oppure audio/video è

possibile registrare una chiamata secondo le modalità riportate in figura 41, nonché visualizzare le statistiche sulla comunicazione corrente (diagramma in fig. 42 a pagina 75).

10.6.1 Diagramma di attività Comunicazione audio

La comunicazione audio (fig. 37) prevede la possibilità svolgimento di più azioni in parallelo, in particolare la registrazione, la condivisione di risorse, una comunicazione testuale e la visualizzazione delle statistiche sulla comunicazione in corso. Inoltre, a discrezione dell'utente, una comunicazione audio può essere promossa in una comunicazione audio/video oppure può essere estesa a nuovi partecipanti.

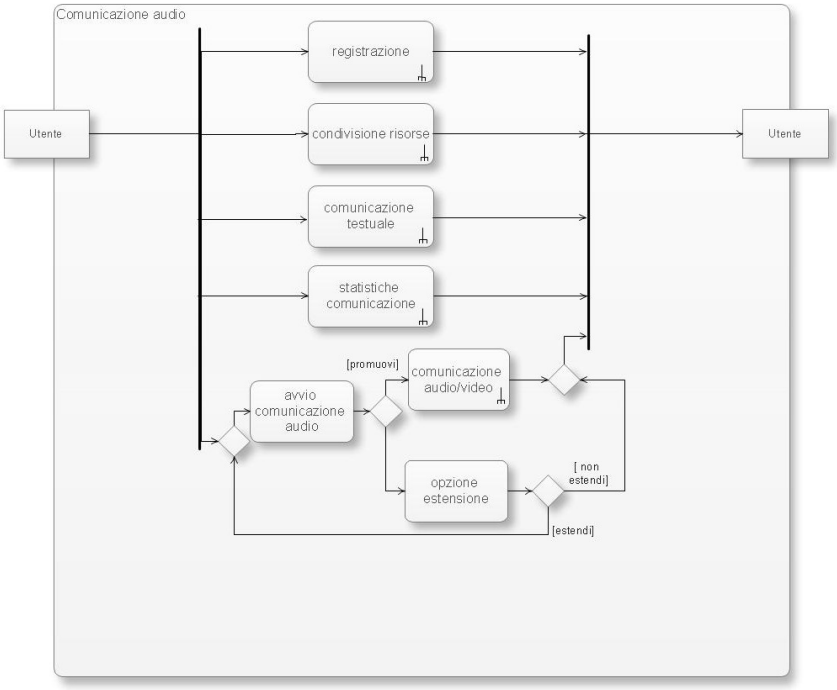

Figura 37: Diagramma di attività relativo alla comunicazione audio

10.6.2 Diagramma di attività Comunicazione audio/video

La comunicazione audio/video (fig. 38 nella pagina successiva) prevede la possibilità svolgimento di più azioni in parallelo, in particolare la registrazione, la condivisione di risorse, una comunicazione testuale e la visualizzazione delle statistiche sulla comunicazione in corso. Inoltre, a discrezione dell'utente, una comunicazione audio/video può essere declassata in una comunicazione audio semplice oppure può essere estesa a nuovi partecipanti.

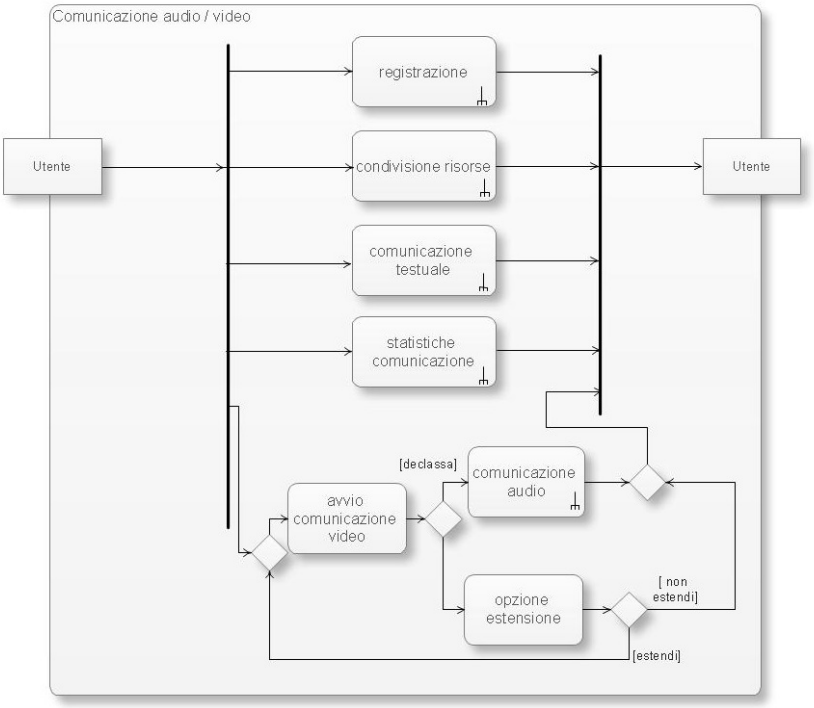

Figura 38: Diagramma di attività relativo alla comunicazione audio/video

10.6.3 Diagramma di attività Comunicazione testuale

La comunicazione testuale, come illustrato dal diagramma riportato in figura 39 nella pagina seguente può avvenire in parallelo con una condivisione di risorse, può essere promossa a comunicazione audio o audio/video e può essere estesa a più partecipanti. Non è prevista, durante una comunicazione testuale, la possibilità di visualizzare statistiche sulla comunicazione o di registrare la stessa in forma persistente.

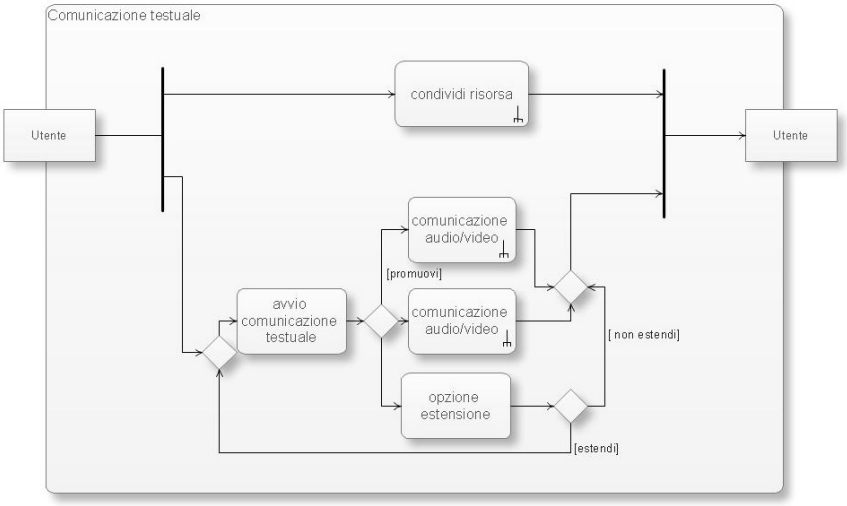

Figura 39: Diagramma di attività relativo alla comunicazione testuale

10.6.4 Diagramma di attività Condivisione di risorse

La sotto-attività di condivisione (fig. 39) di risorse comporta la possibilità di scegliere se condividere un file PDF, un file oppure lo schermo con gli altri partecipanti ad una comunicazione audio, audio/video oppure testuale.

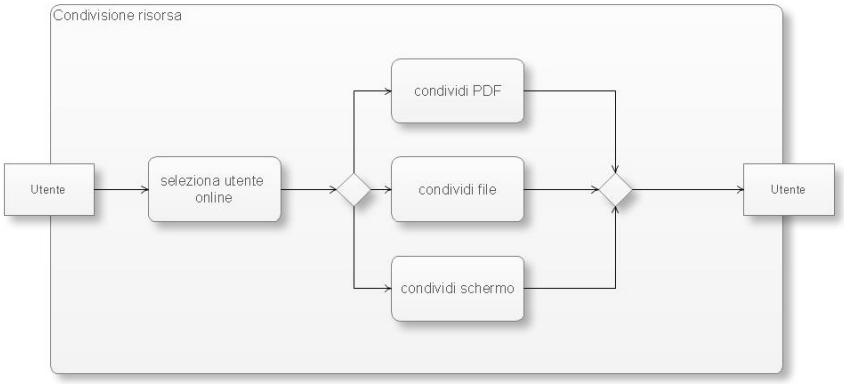

Figura 40: Diagramma di attività relativo alla condivisione di risorse

10.6.5 Diagramma di attività Registrazione chiamata

La registrazione di una chiamata (fig. 41 nella pagina seguente), sotto-attività cui è possibile accedere solo durante una comunicazione audio oppure audio/video, comporta innanzitutto l'inoltro di una richiesta di registrazione agli altri partecipanti della chiamata. Se si ottiene il consenso da parte di tutti, la registrazione ha inizio e prosegue fino a che non viene esplicitamente terminata. Se invece almeno uno degli utenti coinvolti non dà il proprio consenso alla registrazione, questa non può avere luogo.

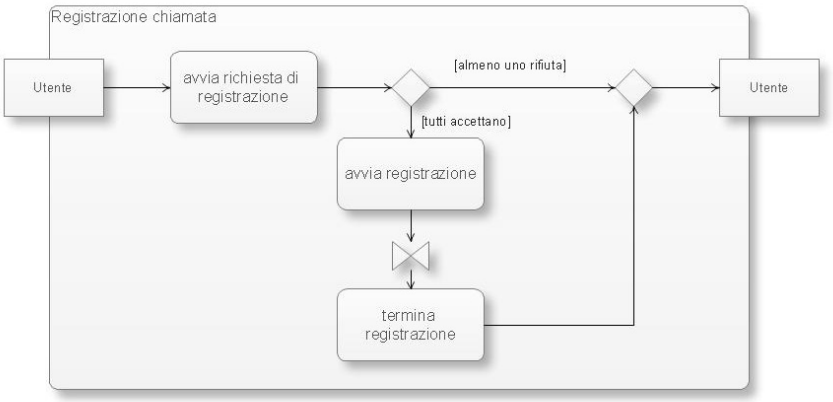

Figura 41: Diagramma di attività relativo alla registrazione della chiamata

10.6.6 Diagramma di attività Statistiche comunicazione

Nel corso di una comunicazione multimediale è possibile visualizzare una serie di informazioni (fig. 42)fra cui:

- il numero di byte inviati e ricevuti;
- la velocità di trasmissione;
- la latenza della connessione;
- il numero di fps (nel caso delle sole comunicazioni video).

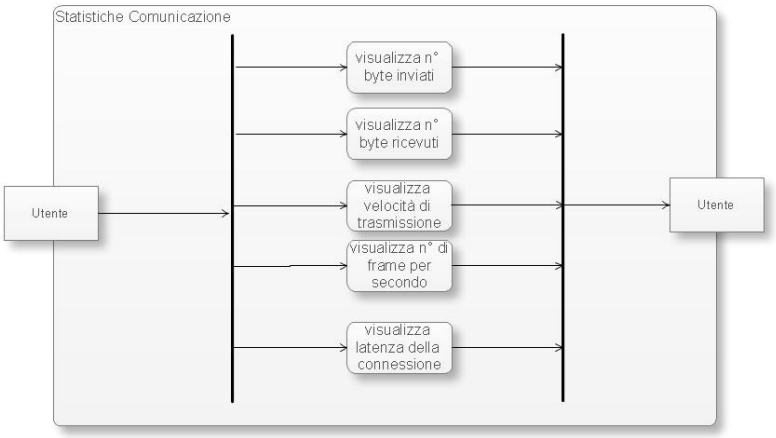

Figura 42: Diagramma di attività relativo alla visualizzazione delle statistiche della comunicazione

10.6.7 Diagramma di attività Messaggio in segreteria

In conclusione, si ricorda che è possibile lasciare un messaggio nella segreteria di un determinato contatto se non presente in linea nel momento in cui desideriamo comunicare con lui 43. I

messaggi che possono essere lasciati in segreteria possono essere costituiti da una sola traccia audio oppure da una traccia audio e da una traccia video.

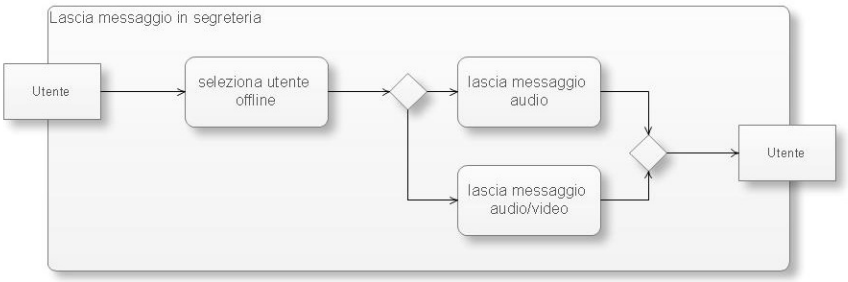

Figura 43: Diagramma di attività relativo alla memorizzazione di un messaggio in segreteria

11 Tracciamenti

Nella seguente sezione vengono proposti tutti i tracciamenti eseguiti mediante il sistema SynthesisRequirementManager. I tracciamenti proposti sono giustificati dalle seguenti due motivazioni:

- Dimostrare il soddisfacimento per necessità e sufficienza della corrispondenza tra gli elementi tracciati (e.g. un componente deve rispondere necessariamente alle esigenze di uno o più requisiti, tali insomma che ne giustifichino l'esistenza. D'altro canto è richiesto che ogni requisito definito in fase d'analisi sia soddisfatto e risolto da almeno un componente).
- dare una lettura generale delle varie: componenti, requisiti, *design pattern* e classi.

11.1 Tracciamenti Requisiti-Componenti

Requisiti	Componenti
RUFO1.0.0	CS07 – Façade del server
	CS02 – Gestione connessione
	CS01 – Gestione database
	CS04 – Gestione autenticazione
RUFD1.1.0	CS01 – Gestione database
RUFD1.1.2	CS01 – Gestione database
RSQO1.2.0	CS02 – Gestione connessione
	CS01 – Gestione database
	CS04 – Gestione autenticazione
RUFO2.0.0	CS01 – Gestione database
	CS07 – Façade del server
	CS04 – Gestione autenticazione
RSQO2.1.0	CS01 – Gestione database
	CS04 – Gestione autenticazione
RSDD2.2.0	CS01 – Gestione database
RUFF3.0.0	CS07 – Façade del server
	CS01 – Gestione database
	CS04 – Gestione autenticazione
RUFF3.1.0	CS01 – Gestione database
	CS04 – Gestione autenticazione
RUFF3.2.0	CS01 – Gestione database
RUFF4.0.0	CS03 – Gestione rubrica
	CS07 – Façade del server
	CS01 – Gestione database
RUFF4.1.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.2.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.3.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.4.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.4.1	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.4.2	CS03 – Gestione rubrica
	CS01 – Gestione database
RUFF4.4.3	CS01 – Gestione database

	CS03 – Gestione rubrica
RUFF4.4.4	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.5.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFF4.6.0	CS03 – Gestione rubrica
RUFF4.7.0	CS01 – Gestione database
	CS03 – Gestione rubrica
RUFO5.0.0	CS01 – Gestione database
	CS07 – Façade del server
RUFF5.1.0	CS01 – Gestione database
RSDO6.0.0	CS02 – Gestione connessione
RUFO6.1.0	CS02 – Gestione connessione
	CS07 – Façade del server
	CS06 – Gestione chiamata
RUFO6.1.1	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF6.1.2	CS02 – Gestione connessione
RUFO6.1.3	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF6.1.4	CS02 – Gestione connessione
RUFO6.2.0	CS07 – Façade del server
	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO6.2.1	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF6.2.2	CS02 – Gestione connessione
RUFO6.2.3	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF6.2.4	CS02 – Gestione connessione
RUFF6.2.5	CS02 – Gestione connessione
RUFF6.3.0	CS02 – Gestione connessione
RUFO6.4.0	CS06 – Gestione chiamata
RUFO 6.5.0	CS06 – Gestione chiamata
RUFO7.0.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO8.0.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO8.1.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO8.2.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO9.0.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO9.1.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFO9.2.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF9.3.0	CS02 – Gestione connessione
RSFO11.0.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RSFO11.1.0	CS02 – Gestione connessione

	CS06 – Gestione chiamata
RSFF11.2.0	CS02 – Gestione connessione
RSFO12.0.0	CS02 – Gestione connessione
	CS07 – Façade del server
	CS06 – Gestione chiamata
RSFF12.1.0	CS02 – Gestione connessione
RUFD12.2.0	CS02 – Gestione connessione
RUFO12.3.0	CS02 – Gestione connessione
	CS06 – Gestione chiamata
RUFF13.0.0	CS07 – Façade del server
	CS02 – Gestione connessione
RUFD13.1.0	CS02 – Gestione connessione
RUFF13.2.0	CS02 – Gestione connessione
RUFF14.0.0	CS02 – Gestione connessione
RUFF14.1.0	CS02 – Gestione connessione
RUFF14.2.0	CS02 – Gestione connessione
RUFF15.0.0	CS05 – Gestione rubrica
	CS01 – Gestione database
	CS02 – Gestione connessione
	CS07 – Façade del server
RUFF15.1.0	CS05 – Gestione rubrica
	CS01 – Gestione database
RUFF15.2.0	CS05 – Gestione rubrica
	CS02 – Gestione connessione
	CS01 – Gestione database
RUFF15.3.0	CS05 – Gestione rubrica
	CS02 – Gestione connessione
	CS01 – Gestione database
RUFF15.4.0	CS01 – Gestione database
	CS05 – Gestione rubrica
	CS02 – Gestione connessione
RUFF15.5.0	CS05 – Gestione rubrica
	CS02 – Gestione connessione
	CS01 – Gestione database
	CS02 – Gestione connessione
	CS07 – Façade del server
	CS02 – Gestione connessione
	CS07 – Façade del server
RUFF18.0.0	CS07 – Façade del server
	CS01 – Gestione database
	CS02 – Gestione connessione
RUFF19.0.0	CS02 – Gestione connessione
RUFF20.0.0	CS02 – Gestione connessione
RUFF20.1.0	CS02 – Gestione connessione
RUFF20.2.0	CS02 – Gestione connessione
RUFF20.3.0	CS02 – Gestione connessione
RUFF20.4.0	CS02 – Gestione connessione

11.2 Tracciamenti Componenti-Requisiti

Componenti	Requisiti associati
CS01 – Gestione database	RUFD1.1.0
	RUFF18.0.0
	RUFF4.4.0
	RUFO1.0.0
	RUFD1.1.2
	RUFF3.0.0
	RUFF4.4.1
	RUFO2.0.0
	RUFO5.0.0
	RUFF15.0.0
	RUFF3.1.0
	RUFF4.4.2
	RUFF15.1.0
	RUFF3.2.0
	RUFF4.4.3
	RUFF15.2.0
	RUFF4.0.0
	RUFF4.4.4
	RSDD2.2.0
	RUFF15.3.0
	RUFF4.1.0
	RUFF4.5.0
	RSQO1.2.0
	RUFF15.4.0
	RUFF4.2.0
	RUFF4.7.0
	RSQO2.1.0
	RUFF15.5.0
	RUFF4.3.0
	RUFF5.1.0
CS02 – Gestione connessione	RSFO12.0.0
	RUFF14.2.0
	RUFF18.0.0
	RUFF6.1.4
	RUFO6.1.0
	RUFO8.1.0
	RSQO1.2.0
	RUFF15.0.0
	RUFF19.0.0
	RUFF6.2.2
	RUFO6.1.1
	RUFO8.2.0
	RUFD12.2.0
	RUFF15.2.0
	RUFF20.0.0
	RUFF6.2.4
	RUFO6.1.3
	RUFO9.0.0
	RSDO6.0.0
	RUFD13.1.0

	RUFF15.3.0
	RUFF20.1.0
	RUFF6.2.5
	RUFO6.2.0
	RUFO9.1.0
	RSFF11.2.0
	RUFF13.0.0
	RUFF15.4.0
	RUFF20.2.0
	RUFF6.3.0
	RUFO6.2.1
	RUFO9.2.0
	RSFF12.1.0
	RUFF13.2.0
	RUFF15.5.0
	RUFF20.3.0
	RUFF9.3.0
	RUFO6.2.3
	RSFO11.0.0
	RUFF14.0.0
	RUFF16.0.0
	RUFF20.4.0
	RUFO1.0.0
	RUFO7.0.0
	RSFO11.1.0
	RUFF14.1.0
	RUFF17.0.0
	RUFF6.1.2
	RUFO12.3.0
	RUFO8.0.0
CS03 – Gestione rubrica	RUFF4.3.0
	RUFF4.7.0
	RUFF4.4.0
	RUFF4.4.1
	RUFF4.4.2
	RUFF4.4.3
	RUFF4.0.0
	RUFF4.4.4
	RUFF4.1.0
	RUFF4.5.0
	RUFF4.2.0
	RUFF4.6.0
CS04 – Gestione autenticazione	RUFO1.0.0
	RUFO1.2.0
	RUFO2.0.0
	RSFD2.1.2
	RUFF3.0.0
	RUFF3.1.0
CS05 – Gestione rubrica	RUFF15.5.0
	RUFF15.0.0
	RUFF15.1.0
	RUFF15.2.0

	RUFF15.3.0
	RUFF15.4.0
CS06 – Gestione chiamata	RUFO6.1.0
	RUFO6.1.1
	RUFO6.1.3
	RUFO6.2.0
	RUFO6.2.1
	RUFO6.2.3
	RUFO6.4.0
	RUFO6.5.0
	RUFO7.0.0
	RUFO8.0.0
	RUFO8.1.0
	RUFO8.2.0
	RUFO9.0.0
	RUFO9.1.0
	RUFO9.2.0
	RSFO11.0.0
	RSFO11.1.0
	RSFO12.0.0
	RUFO12.3.0
CS07 – Façade del server	RUFF4.0.0
	RSFO12.0.0
	RUFO1.0.0
	RUFF13.0.0
	RUFO2.0.0
	RUFF15.0.0
	RUFO5.0.0
	RUFF16.0.0
	RUFO6.1.0
	RUFF17.0.0
	RUFO6.2.0
	RUFF18.0.0
	RUFF3.0.0
CP01 – Gestione comunicazione	RUFO6.1.0
	RUFO6.1.1
	RUFO6.1.3
	RUFO6.2.0
	RUFO6.2.1
	RUFO6.2.3
	RUFO6.4.0
	RUFO6.5.0
	RUFO7.0.0
	RUFO8.0.0
	RUFO8.1.0
	RUFO8.2.0
	RUFO9.0.0
	RUFO9.1.0
	RUFO9.2.0
	RUFF6.1.2
	RUFF6.1.4
	RUFF6.2.2

	RUFF6.2.4
	RUFF6.2.5
	RUFF6.3.0
	RSDO6.0.0
CP02 – Rappresentazione dati	RUFO1.0.0
	RUFO1.2.0
	RUFO2.1.0
	RUFF3.0.0
	RUFF3.1.0
	RUFF3.2.0
CP03 – Gestione GUI	RSFD21.0.0
	RSQF26.0.0
	RSDO10.0.0
	RSDO10.1.0
	RSFD21.0.0
CV02 – Login	RUFO1.0.0
	RSFO1.2.0
	RSFO2.1.0
	RSDD2.2.0

11.3 Tracciamenti Componenti-DesignPattern

Componenti	Design pattern utilizzati
CP03 – Gestione GUI	MVP
CS01 – Gestione database	Data Access Object
	Singleton
	MVP
CS02 – Gestione connessione	Factory Method
	Singleton
CP01 – Gestione comunicazione	Singleton
CS06 – Gestione chiamate	MVP
CS05 – Gestione rubrica	MVP
CS04 – Gestione autenticazione	Strategy
CS03 – Gestione rubrica	MVP
CS07 – Façade del server	Façade
	MVP

11.4 Tracciamenti DesignPattern-Componenti

Design pattern	Componenti
Data Access Object	CS01 – Gestione database
Façade	CS07 – Façade del server
Factory Method	CS02 – Gestione connessione
Singleton	CS02 – Gestione connessione
Strategy	CS04 – Gestione autenticazione

MVP	CS01 – Gestione database
	CS03 – Gestione rubrica
	CS05 – Gestione rubrica
	CP01 – Gestione comunicazione
	CS06 – Gestione chiamate
	CP03 – Gestione GUI

11.5 Tracciamenti Componenti-Classi

Componenti	Classi
CS01 – Gestione database	server.dao.CallDAO
	server.dao.CallListDAO
	server.dao.GroupDAO
	server.dao.MessageDAO
	server.dao.UserDataDAO
	server.dao.AddressBookEntryDAO
	server.dao.HibernateUtil
CS03 – Gestione rubrica	server.abook.AddressBookEntry
	server.abook.IAddressBookEntry
	server.abook.IGroup
	server.abook.Group
	server.abook.IUserData
	server.abook.UserData
CS05 – Gestione rubrica	server.message.IMessage
	server.message.Message
CS06 – Gestione chiamate	server.call.ICall
	server.call.Call
	server.call.ICallList
	server.call.CallList
CS02 – Gestione connessione	server.connection.PushInbound
	org.apache.catalina.websocket.MessageInbound
CS04 – Gestione autenticazione	server.authentication.AuthenticationModule
	server.authentication.CredentialLoader
	server.authentication.PrincipalImpl
	server.authentication.IAuthenticationData
	server.authentication.AuthenticationData
	server.authentication.AESAlgorithm
	server.authentication.ISecurityStrategy
	javax.security.auth.spi.LoginModule
	javax.security.auth.callback.CallbackHandler
	javax.security.Principal
CS07 – Façade del server	server.connection.ChannelServlet
	server.abook.servlet.AddressBookDoAddContactServlet
	server.abook.servlet.AddressBookDoRemoveContactServlet
	server.abook.servlet.AddressBookDoCreateGroupServlet
	server.abook.servlet.AddressBookDoDeletGroupServlet
	server.abook.servlet.AddressBookDoInsertInGroupServlet
	server.abook.servlet.AddressBookDoRemoveInGroupServlet
	server.abook.servlet.AddressBookDoBlockServlet

	server.abook.servlet.AddressBookDoUnblockServlet
	server.abook.servlet.AddressBookGetContactsServlet
	server.abook.servlet.AddressBookGetGroupsServlet
	server.abook.servlet.AddressBookDoSearchServlet
	server.authentication.servlet.LoginServlet
	server.authentication.servlet.LogoutServlet
	server.authentication.servlet.RegisterServlet
	server.message.servlet.InsertMessageServlet
	server.message.servlet.DeletMessageServlet
	server.message.servlet.UpdateStatusMessageServlet
	server.message.servlet.DownloadMessageListServlet
	server.call.servlet.DownloadCallHistoryManager
	javax.servlet.http.HttpServlet
	org.apache.catalina.websocket.WebSocketServlet
CP01 – Gestione comunicazione	clientpresenter.kernel.CommunicationCenter
	PeerICECandidate
	WebKitRTCPeerConnection
	PeerSessionDescription
CP02 – Rappresentazione dati	clientpresenter.data.JSCall
	clientpresenter.data.JSGroup
	clientpresenter.data.JSMessage
	clientpresenter.data.JSUserData
CP03 – Gestione GUI	clientpresenter.guicontrol.AccountSettingsPanelPresenter
	clientpresenter.guicontrol.AddressBookPanelPresenter
	clientpresenter.guicontrol.CallHistoryPanelPresenter
	clientpresenter.guicontrol.CommunicationPanelPresenter
	clientpresenter.guicontrol.GroupPanelPresenter
	clientpresenter.guicontrol.SearchResultPanelPresenter
	clientpresenter.guicontrol.ContactPanelPresenter
	clientpresenter.guicontrol.LoginPanelPresenter
	clientpresenter.guicontrol.RegisterPanelPresenter
	clientpresenter.guicontrol.TopLevelPresenter
	clientpresenter.guicontrol.ChildPrenter
	clientpresenter.guicontrol.MainPanelPresenter
	clientpresenter.guicontrol.MessagePanelPresenter
	clientpresenter.guicontrol.PresenterMediator
	clientpresenter.guicontrol.ToolsPanelPresenter
CV01 – GUI	clientview.MainPanel
	clientview.ToolsPanel
	clientview.AddressBookPanel
	clientview.ContactPanel
	clientview.MessagePanel
	clientview.GroupPanel
	clientview.SearchPanel
	clientview.AccountSettingsPanel
	clientview.CallHistoryPanel
	clientview.CommunicationPanel
CV02 – Login	clientview.LoginPanel
	clientview.RegisterPanel

11.6 Tracciamenti Classi-Componenti

Classi	Componenti
clientpresenter.data.JSCall	CP02 – Rappresentazione dati
clientpresenter.data.JSGroup	CP02 – Rappresentazione dati
clientpresenter.data.JSMessage	CP02 – Rappresentazione dati
clientpresenter.data.JSUserData	CP02 – Rappresentazione dati
clientpresenter.guicontrol.AccountSettingsPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.AddressBookPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.CallHistoryPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.ChildPrenter	CP03 – Gestione GUI
clientpresenter.guicontrol.CommunicationPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.ContactPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.GroupPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.LoginPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.MainPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.MessagePanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.PresenterMediator	CP03 – Gestione GUI
clientpresenter.guicontrol.RegisterPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.SearchResultPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.ToolsPanelPresenter	CP03 – Gestione GUI
clientpresenter.guicontrol.TopLevelPresenter	CP03 – Gestione GUI
clientpresenter.kernel.CommunicationCenter	CP01 – Gestione comunicazione
clientview.AccountSettingsPanel	CV01 – GUI
clientview.AddressBookPanel	CV01 – GUI
clientview.CallHistoryPanel	CV01 – GUI
clientview.CommunicationPanel	CV01 – GUI
clientview.ContactPanel	CV01 – GUI
clientview.GroupPanel	CV01 – GUI
clientview.LoginPanel	CV02 – Login
clientview.MainPanel	CV01 – GUI
clientview.MessagePanel	CV01 – GUI
clientview.RegisterPanel	CV02 – Login
clientview.SearchPanel	CV01 – GUI
clientview.ToolsPanel	CV01 – GUI
javax.security.auth.callback.CallbackHandler	CS04 – Gestione autenticazione
javax.security.auth.spi.LoginModule	CS04 – Gestione autenticazione
javax.security.Principal	CS04 – Gestione autenticazione
javax.servlet.http.HttpServlet	CS07 – Façade del server
org.apache.catalina.websocket.MessageInbound	CS02 – Gestione connessione
org.apache.catalina.websocket.WebSocketServlet	CS07 – Façade del server
PeerICECandidate	CP01 – Gestione comunicazione
PeerSessionDescription	CP01 – Gestione comunicazione
server.abook.AddressBookEntry	CS03 – Gestione rubrica
server.abook.Group	CS03 – Gestione rubrica
server.abook.IAddressBookEntry	CS03 – Gestione rubrica
server.abook.IGroup	CS03 – Gestione rubrica
server.abook.IUserData	CS03 – Gestione rubrica

server.abook.servlet.AddressBookDoAddContactServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoBlockServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoCreateGroupServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoDeletGroupServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoInsertInGroupServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoRemoveContactServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoRemoveInGroupServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoSearchServlet	CS07 – Façade del server
server.abook.servlet.AddressBookDoUnblockServlet	CS07 – Façade del server
server.abook.servlet.AddressBookGetContactsServlet	CS07 – Façade del server
server.abook.servlet.AddressBookGetGroupsServlet	CS07 – Façade del server
server.abook.UserData	CS03 – Gestione rubrica
server.authentication.AESAlgorithm	CS04 – Gestione autenticazione
server.authentication.AuthenticationData	CS04 – Gestione autenticazione
server.authentication.AuthenticationModule	CS04 – Gestione autenticazione
server.authentication.CredentialLoader	CS04 – Gestione autenticazione
server.authentication.IAuthenticationData	CS04 – Gestione autenticazione
server.authentication.ISecurityStrategy	CS04 – Gestione autenticazione
server.authentication.PrincipalImpl	CS04 – Gestione autenticazione
server.authentication.servlet.LoginServlet	CS07 – Façade del server
server.authentication.servlet.LogoutServlet	CS07 – Façade del server
server.authentication.servlet.RegisterServlet	CS07 – Façade del server
server.dao.CallDAO	CS01 – Gestione database
server.dao.CallListDAO	CS01 – Gestione database
server.call.Call	CS06 – Gestione chiamate
server.call.ICall	CS06 – Gestione chiamate
server.call.CallList	CS06 – Gestione chiamate
server.call.ICallList	CS06 – Gestione chiamate
server.call.servlet.DownloadCallHistoryManager	CS07 – Façade del server
server.connection.ChannelServlet	CS07 – Façade del server
server.connection.PushInbound	CS02 – Gestione connessione
server.dao.AddressBookEntryDAO	CS01 – Gestione database
server.dao.GroupDAO	CS01 – Gestione database
server.dao.HibernateUtil	CS01 – Gestione database
server.dao.MessageDAO	CS01 – Gestione database
server.dao.UserDataDAO	CS01 – Gestione database
server.message.IMessage	CS05 – Gestione rubrica
server.message.Message	CS05 – Gestione rubrica
server.message.servlet.DeletMessageServlet	CS07 – Façade del server
server.message.servlet.DownloadMessageListServlet	CS07 – Façade del server
server.message.servlet.InsertMessageServlet	CS07 – Façade del server
server.message.servlet.UpdateStatusMessageServlet	CS07 – Façade del server
WebKitRTCPeerConnection	CP01 – Gestione comunicazione