



# MyTalk

## Specifica tecnica

---

### Informazioni sul documento

---

<b>Nome file:</b>	specifiche_tecniche.1.0.pdf
<b>Versione:</b>	1.0
<b>Data creazione:</b>	2013-01-23
<b>Data ultima modifica:</b>	2013-01-30
<b>Stato:</b>	Approvato
<b>Uso:</b>	Esterno
<b>Lista di distribuzione:</b>	Prof. Tullio Vardanega Prof. Riccardo Cardin Dott. Gregorio Piccoli Team SoftwareSynthesis
<b>Redattori:</b>	Diego Beraldin Elena Zecchinato Marco Schivo
<b>Approvato da:</b>	Riccardo Tresoldi
<b>Verificatori:</b>	Andrea Meneghinello Stefano Farronato

---

## Storia delle modifiche

Versione	Descrizione intervento	Membro	Ruolo	Data
1.0	Approvazione documento	Riccardo Tresoldi	Responsabile	2012-01-30
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

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Scopo del prodotto . . . . .	1
1.2	Scopo del documento . . . . .	1
1.3	Glossario . . . . .	1
<b>2</b>	<b>Riferimenti</b>	<b>2</b>
2.1	Normativi . . . . .	2
2.2	Informativi . . . . .	2
<b>3</b>	<b>Strumenti utilizzati</b>	<b>3</b>
3.1	Java . . . . .	3
3.2	Hibernate . . . . .	3
<b>4</b>	<b>Design Pattern utilizzati</b>	<b>4</b>
4.1	Composite . . . . .	4
4.1.1	Scopo . . . . .	4
4.1.2	Diagramma esemplificativo . . . . .	5
4.1.3	Componenti che lo implementano . . . . .	5
4.2	Data Access Object (DAO) . . . . .	5
4.2.1	Scopo . . . . .	5
4.2.2	Diagramma esemplificativo . . . . .	6
4.2.3	Componenti che lo implementano . . . . .	6
4.3	Façade . . . . .	6
4.3.1	Scopo . . . . .	6
4.3.2	Diagramma esemplificativo . . . . .	7
4.3.3	Componenti che lo implementano . . . . .	7
4.4	Factory Method . . . . .	7
4.4.1	Scopo . . . . .	7
4.4.2	Diagramma esemplificativo . . . . .	8
4.4.3	Componenti che lo implementano . . . . .	8
4.5	Model-View-Presenter . . . . .	8
4.5.1	Scopo . . . . .	8
4.5.2	Diagrammi esemplificativi . . . . .	8
4.5.3	Componenti che lo implementano . . . . .	9
4.6	Observer . . . . .	10
4.6.1	Scopo . . . . .	10
4.6.2	Diagramma esemplificativo . . . . .	10
4.6.3	Componenti che lo implementano . . . . .	10
4.7	Proxy . . . . .	10
4.7.1	Scopo . . . . .	10
4.7.2	Diagramma esemplificativo . . . . .	11
4.7.3	Componenti che lo implementano . . . . .	11
4.8	Singleton . . . . .	11
4.8.1	Scopo . . . . .	11
4.8.2	Diagramma esemplificativo . . . . .	11
4.8.3	Componenti che lo implementano . . . . .	12
4.9	State . . . . .	12
4.9.1	Scopo . . . . .	12
4.9.2	Diagramma esemplificativo . . . . .	12
4.9.3	Componenti che lo implementano . . . . .	12

<b>5 Introduzione all'architettura di sistema</b>	<b>14</b>
<b>6 Architettura del database</b>	<b>16</b>
6.1 Progettazione concettuale . . . . .	16
6.1.1 Lista delle classi . . . . .	16
6.1.2 Gerarchia tra classi . . . . .	17
6.1.3 Associazione tra classi . . . . .	17
6.1.4 Diagramma delle classi . . . . .	18
6.2 Progettazione logica . . . . .	18
6.2.1 Rappresentazione delle gerarchie . . . . .	18
6.2.2 Rappresentazione delle associazioni . . . . .	18
6.2.3 Lista delle classi . . . . .	19
6.2.4 Diagramma delle classi . . . . .	20
<b>7 Architettura mytalk.server</b>	<b>21</b>
7.1 Componenti evidenziati . . . . .	21
7.1.1 Gestione database . . . . .	21
7.1.2 Gestione connessione . . . . .	22
7.1.3 Gestione rubrica . . . . .	22
7.1.4 Gestione stato . . . . .	24
7.1.5 Gestione segreteria . . . . .	25
7.1.6 Façade del server . . . . .	26
7.2 Diagramma delle classi . . . . .	27
<b>8 Architettura mytalk.clientpresenter</b>	<b>28</b>
8.1 Logica di rete . . . . .	28
8.1.1 L'idea iniziale . . . . .	29
8.2 Componenti evidenziati . . . . .	29
8.2.1 Gestione comunicazione . . . . .	29
8.2.2 Façade del presenter . . . . .	30
8.3 Diagramma delle classi . . . . .	31
<b>9 Architettura mytalk.clientview</b>	<b>32</b>
9.1 Componenti evidenziati . . . . .	33
9.1.1 Façade della vista . . . . .	33
9.1.2 Gestione GUI . . . . .	33
9.2 Diagramma delle classi . . . . .	35
<b>10 Descrizione delle classi</b>	<b>36</b>
10.1 Package org.softwaresynthesis.mytalk.server.dao . . . . .	36
10.1.1 IAudioMessage . . . . .	36
10.1.2 IAudioVideoMessage . . . . .	36
10.1.3 AudioMessage . . . . .	36
10.1.4 AudioVideoMessage . . . . .	36
10.1.5 IGroup . . . . .	36
10.1.6 StandardGroup . . . . .	37
10.1.7 IUserData . . . . .	37
10.1.8 StandardUserData . . . . .	37
10.2 Package org.softwaresynthesis.mytalk.server.connection . . . . .	37
10.2.1 ICommunicationHandler . . . . .	37
10.2.2 StandardCommunicationHandler . . . . .	37
10.2.3 IConnection . . . . .	38
10.2.4 WebRTCInfo . . . . .	38

10.3	Package org.softwaresynthesis.mytalk.server.abook . . . . .	38
10.3.1	IContact . . . . .	38
10.3.2	IAddressBook . . . . .	38
10.3.3	UserDataProxy . . . . .	38
10.3.4	AddressBook . . . . .	39
10.4	Package org.softwaresynthesis.mytalk.server.state . . . . .	39
10.4.1	IState . . . . .	39
10.4.2	StateOnline . . . . .	39
10.4.3	StateOffline . . . . .	39
10.4.4	StateAvailable . . . . .	39
10.4.5	StateOccupied . . . . .	40
10.5	Package org.softwaresynthesis.mytalk.server.message . . . . .	40
10.5.1	IMessageBox . . . . .	40
10.5.2	StandardMessageBox . . . . .	40
10.5.3	AudioMessageProxy . . . . .	40
10.5.4	AudioVideoMessageProxy . . . . .	40
10.6	Package org.softwaresynthesis.mytalk.server . . . . .	41
10.6.1	IServerFacade . . . . .	41
10.6.2	StandardServerFacade . . . . .	41
10.7	Package org.softwaresynthesis.mytalk.clientpresenter . . . . .	41
10.7.1	IClient . . . . .	41
10.7.2	StandardClient . . . . .	41
10.7.3	IPresenterFacade . . . . .	42
10.7.4	StandardPresenterFacade . . . . .	42
10.8	Package org.softwaresynthesis.mytalk.clientview . . . . .	42
10.8.1	IViewFacade . . . . .	42
10.8.2	StandardViewFacade . . . . .	42
10.9	Package org.softwaresynthesis.clientview.gui . . . . .	42
10.9.1	GUIHandler . . . . .	42
10.9.2	AddressBookPanel . . . . .	43
10.9.3	MainPanel . . . . .	43
10.9.4	ToolsPanel . . . . .	43
10.9.5	SearchResultPanel . . . . .	43
10.9.6	ContactPanel . . . . .	43
10.9.7	MessagePanel . . . . .	44
10.9.8	LanguagePanel . . . . .	44
10.9.9	AccountSettingsPanel . . . . .	44
10.9.10	CallHistoryPanel . . . . .	44
<b>11</b>	<b>Conclusioni sull'architettura</b>	<b>45</b>
11.1	Diagrammi delle attività . . . . .	45
<b>12</b>	<b>Tracciamenti</b>	<b>56</b>
12.1	Tracciamenti Requisiti-Componenti . . . . .	56
12.2	Tracciamenti Componenti-Requisiti . . . . .	58
12.3	Tracciamenti Componenti-DesignPattern . . . . .	60
12.4	Tracciamenti DesignPattern-Componenti . . . . .	61
12.5	Tracciamenti Componenti-Classi . . . . .	61
12.6	Tracciamenti Classi-Componenti . . . . .	63

## Elenco delle figure

1	Diagramma ad alto livello del pattern Composite . . . . .	5
2	Diagramma ad alto livello del pattern Data Access Object . . . . .	6
3	Diagramma ad alto livello del pattern Façade . . . . .	7
4	Diagramma ad alto livello del pattern Factory Method . . . . .	8
5	Diagramma ad alto livello del pattern MVP . . . . .	8
6	Diagramma di sequenza che illustra le collaborazioni in MVP . . . . .	9
7	Diagramma ad alto livello del pattern Observer . . . . .	10
8	Diagramma ad alto livello del pattern Proxy . . . . .	11
9	Diagramma ad alto livello del pattern Singleton . . . . .	11
10	Diagramma ad alto livello del pattern State . . . . .	12
11	Diagramma dei Package - Generale . . . . .	15
12	Diagramma delle classi - Schema concettuale database . . . . .	18
13	Diagramma delle classi - Schema logico Database . . . . .	20
14	Diagramma delle classi - Gestione database . . . . .	21
15	Diagramma delle classi - Gestione connessione . . . . .	22
16	Diagramma delle classi - Gestione rubrica . . . . .	23
17	Diagramma delle classi - Gestione stato . . . . .	24
18	Diagramma delle classi - Gestione segreteria . . . . .	25
19	Diagramma delle classi - Façade del server . . . . .	26
20	Diagramma delle classi - Architettura mytalk.server . . . . .	27
21	Diagramma delle classi - Gestione comunicazione . . . . .	29
22	Diagramma delle classi - Façade del presenter . . . . .	30
23	Diagramma delle classi - Architettura mytalk.clientpresenter . . . . .	31
24	Rappresentazione ad alto livello della GUI . . . . .	32
25	Diagramma delle classi - Façade della vista . . . . .	33
26	Diagramma delle classi - Gestione GUI . . . . .	34
27	Diagramma delle classi - Architettura mytalk.clientview . . . . .	35
28	Diagramma di attività generale che descrive l'interazione con il sistema . . . . .	45
29	Diagramma di attività relativo all'autenticazione . . . . .	46
30	Diagramma di attività relativo alla registrazione . . . . .	47
31	Diagramma di attività relativo al recupero della password . . . . .	48
32	Diagramma di attività relativo alla gestione della rubrica personale . . . . .	49
33	Diagramma di attività relativo alla gestione dei dati dell'account personale . . . . .	50
34	Diagramma di attività relativo alla gestione della segreteria . . . . .	50
35	Diagramma di attività relativo alla connessione . . . . .	51
36	Diagramma di attività relativo alla comunicazione audio . . . . .	52
37	Diagramma di attività relativo alla comunicazione audio/video . . . . .	53
38	Diagramma di attività relativo alla comunicazione testuale . . . . .	53
39	Diagramma di attività relativo alla condivisione di risorse . . . . .	54
40	Diagramma di attività relativo alla registrazione della chiamata . . . . .	54
41	Diagramma di attività relativo alla memorizzazione di un messaggio in segreteria	55

## 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 dei requisiti e dei componenti, negli schemi: requisiti-componenti e componenti-requisiti;
- 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.2.0.pdf* dove vengono definiti e descritti tutti i termini marcati con una sottolineatura.

## 2 Riferimenti

### 2.1 Normativi

*piano\_di\_qualifica.2.0.pdf* allegato.

*norme\_di\_progetto.2.0.pdf* allegato.

*analisi\_dei\_requisiti.2.0.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 patterns: *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.2.0.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).

##### Svantaggi

- potenziale lentezza causata dall'utilizzo della JVM e dal fatto di essere un linguaggio interpretato.

#### 3.2 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;
- essendo rilasciato sotto licenza LGPL può essere utilizzato senza restrizioni (copyleft) e vincoli di licenza delle opere derivate.

## 4 Design Pattern utilizzati

In questa sezione discuteremo i design pattern utilizzati nella progettazione dei componenti architetturali. Ogni design pattern sarà proposto con la seguente forma:

- **Scopo:** verrà proposto lo scopo generico del pattern, al fine di evidenziare subito la sua utilità.
- **Diagramma esemplificativo:** si riporterà lo schema UML, rappresentante un'implementazione generica del design pattern in esame.
- **Vantaggi derivanti:** si darà un elenco dei vantaggi apportati dall'utilizzo del pattern, in particolare sotto il profilo della manutenzione e del riuso del codice.
- **Componenti che lo implementano:** infine verranno elencati i componenti dell'architettura di sistema, che implementano il pattern descritto.

Per una visione d'insieme dei componenti utilizzati da un pattern, e dei pattern utilizzati da un componente, rimandiamo alle sottosezioni “Tracciamenti Componenti-Design Pattern” e “Tracciamenti Design Pattern-Componenti” della sezione 12 a pagina 56.

### 4.1 Composite

#### 4.1.1 Scopo

Il pattern Composite ha lo scopo di comporre oggetti in strutture ad albero al fine di rappresentare gerarchie parte-tutto e consentire ai clienti di trattare oggetti singoli e composizioni in modo uniforme. Permette inoltre di gestire strutture dati gerarchicizzate con elementi “foglie” ed elementi “contenitori”, l’ideale per la struttura “gruppo” e “utente”.

### 4.1.2 Diagramma esemplificativo

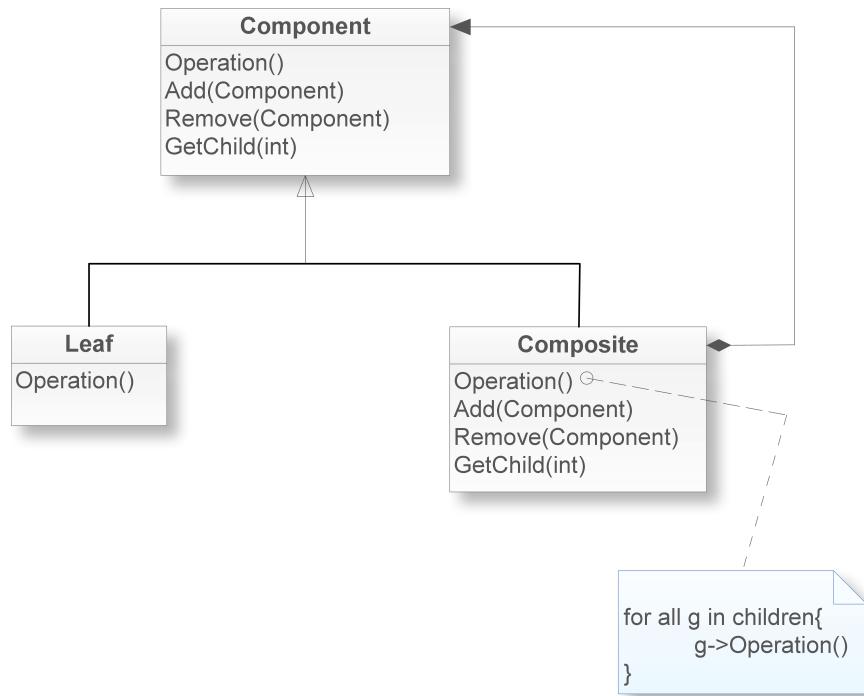


Figura 1: Diagramma ad alto livello del pattern Composite

### 4.1.3 Componenti che lo implementano

#### GESTIONE RUBRICA

Composite permette di trattare in maniera omogenea singoli oggetti e oggetti composti, come gli utenti e gruppi di utenti della rubrica. Inoltre, dal momento che rende più semplice l'aggiunta di componenti, permetterebbe in futuro l'integrazione di nuove tipologie di utenti senza la necessità di modificare la struttura preesistente.

Lo svantaggio principale che comporta l'uso di Composite è la mancanza di limiti nell'aggiunta di nuove tipologie di componenti. Per far fronte a questo rischio si è introdotta la classe `org.softwaresynthesis.mytalk.server.abook.AddressBook` che controlla l'accesso alla struttura dati corrispondente alla rubrica.

## 4.2 Data Access Object (DAO)

### 4.2.1 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 ad una classe DAO rendendo i componenti che implementano la logica di business indipendenti dalla natura del dispositivo di persistenza. Questo approccio garantisce che un eventuale cambiamento del dispositivo di persistenza non comporti modifiche sui componenti di business.

#### 4.2.2 Diagramma esemplificativo

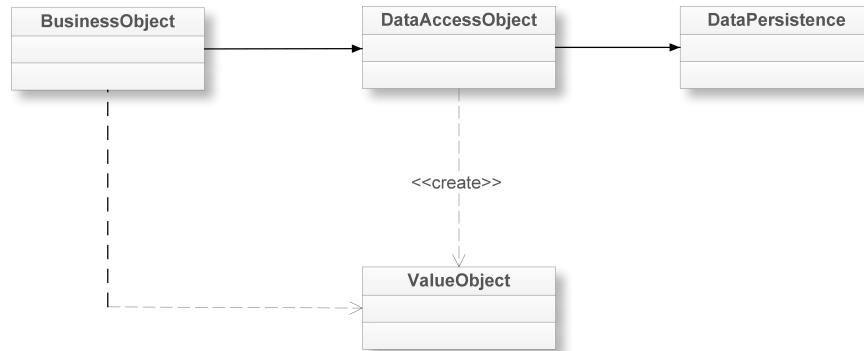


Figura 2: Diagramma ad alto livello del pattern Data Access Object

#### 4.2.3 Componenti che lo implementano

##### GESTIONE DATABASE

Le classi DAO consentono di isolare l'accesso alle tabelle del database dalla parte di business logic facendo corrispondere alle invocazioni di metodo le opportune operazioni sui record del database.

L'utilizzo di tale pattern crea inoltre un maggiore livello di astrazione e mantiene una rigida separazione tra le sotto-architetture corrispondenti a model e presenter.

### 4.3 Façade

#### 4.3.1 Scopo

Fornire un'interfaccia unificata per un insieme di interfacce o classi presenti in una sotto-architettura. Façade definisce inoltre un'interfaccia di livello più alto che rende la sotto-architettura più semplice da utilizzare.

#### 4.3.2 Diagramma esemplificativo

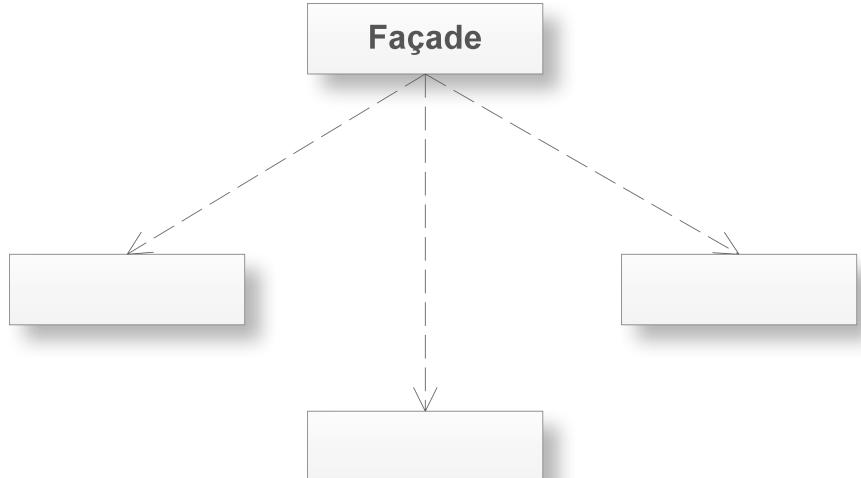


Figura 3: Diagramma ad alto livello del pattern Façade

#### 4.3.3 Componenti che lo implementano

##### **FAÇADE DEL SERVER**

L'uso di Façade permette di esporre verso i client una sorta di interfaccia semplificata nascondendo i componenti della sotto-architettura server, fornendo un punto di accesso centralizzato e riducendo il numero di dipendenze funzionali fra le classi del server e i componenti appartenenti a sotto-architetture esterne.

##### **FAÇADE DEL PRESENTER**

Tramite questo design pattern si introduce un livello di indirezione fra le sotto-architetture clientpresenter e clientview, che risultano dunque indipendenti.

##### **FAÇADE DELLA VISTA**

Data la forte bidirezionalità delle interazioni fra componenti delle sotto-architettura clientpresenter e clientview, è stata introdotta una sorta di facciata anche alla vista in modo da facilitare il tracciamento delle dipendenze fra i componenti del presenter e quelli della view.

### 4.4 Factory Method

#### 4.4.1 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.

#### 4.4.2 Diagramma esemplificativo

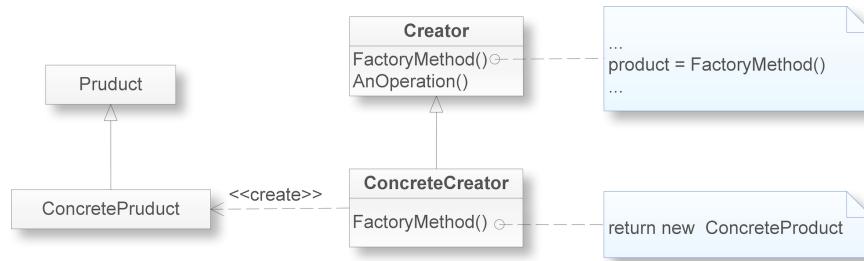


Figura 4: Diagramma ad alto livello del pattern Factory Method

#### 4.4.3 Componenti che lo implementano

##### FAÇADE DEL SERVER

Factory Method permette ai client di ottenere con facilità degli oggetti proxy che specializzano le interfacce `server.dao.IAudioMessage`, `server.dao.IAudioVideoMessage` e `server.dao.IUserData`.

Questo permette di ridurre il traffico di rete in quanto oggetti potenzialmente di grandi dimensioni rimangono sul server e vengono scaricati solo quando se ne presenta l'effettiva necessità.

##### GESTIONE CONNESSIONE

Gli oggetti che rappresentano connessioni, sottotipi di `server.connection.IConnection` sono ottenuti mediante un Factory Method nelle classi concrete che implementano l'interfaccia `server.connectionICommunicationHandler`.

Ciò garantisce maggiore flessibilità in quanto permette in futuro di gestire più categorie di handler che restituiscono diversi tipi connessione.

### 4.5 Model-View-Presenter

#### 4.5.1 Scopo

Il pattern architettonale 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.

#### 4.5.2 Diagrammi esemplificativi

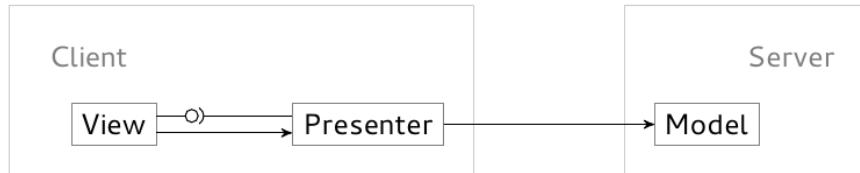


Figura 5: Diagramma ad alto livello del pattern MVP

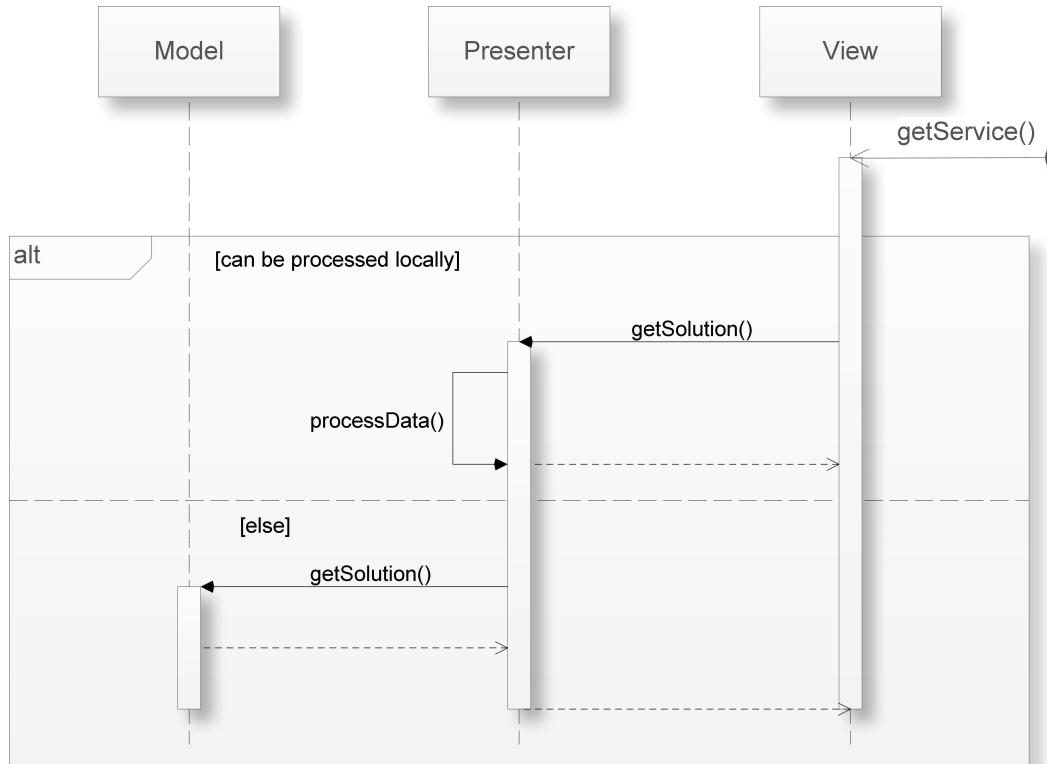


Figura 6: Diagramma di sequenza che illustra le collaborazioni in MVP

Come si evince dal diagramma riportato in figura 6 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.

#### 4.5.3 Componenti che lo implementano

MVP viene utilizzato come il pattern più ad alto livello del nostro 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.

In particolare, le parti del sistema che usano questo pattern corrispondono alle sotto-architetture:

- server
- clientpresenter
- clientview

## 4.6 Observer

### 4.6.1 Scopo

Definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente.

### 4.6.2 Diagramma esemplificativo

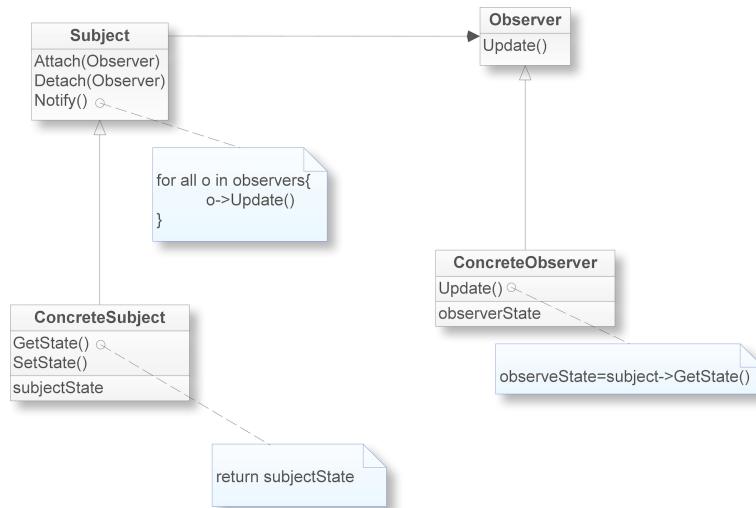


Figura 7: Diagramma ad alto livello del pattern Observer

### 4.6.3 Componenti che lo implementano

#### GESTIONE STATO

Il pattern Observer è utile in quanto permette agli utenti di osservare lo stato degli altri, ricevendo in modo automatico e trasparente una notifica nel caso in cui uno di questi ultimi subisse variazioni, essendo ogni utente sia osservato che osservatore.

Al momento della connessione, infatti, ogni utente si registra come osservatore sui suoi contatti che sono online e, al contempo, li aggiunge tra i propri osservatori. In tal modo gli utenti notificano in broadcast le loro variazioni di stato e sono sempre aggiornati sullo stato dei contatti della loro rubrica.

## 4.7 Proxy

### 4.7.1 Scopo

Fornisce un placeholder per un altro oggetto in modo da controllarne l'accesso e consentire un uso ottimizzato della memoria.

#### 4.7.2 Diagramma esemplificativo

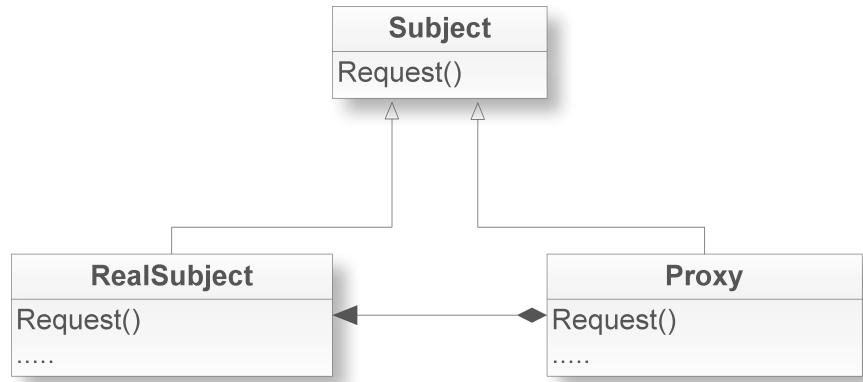


Figura 8: Diagramma ad alto livello del pattern Proxy

#### 4.7.3 Componenti che lo implementano

##### GESTIONE RUBRICA

L'utilizzo di un proxy al posto di un utente permette di raggiungere una maggiore efficienza limitando l'utilizzo della rete evitando all'utente di percepire una eccessiva lentezza che comprometterebbe la sua esperienza.

Inoltre, tramite un proxy è possibile controllare l'accesso ai dati (che risiedono nel server) garantendo dunque un migliore livello di protezione.

##### GESTIONE SEGRETERIA

Poiché i messaggi audio e, soprattutto, i messaggi audio/video possono essere di grandi dimensioni, i proxy permettono di ottimizzare il consumo della memoria e di evitare l'attesa da parte dell'utente se non strettamente necessario.

### 4.8 Singleton

#### 4.8.1 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.

#### 4.8.2 Diagramma esemplificativo

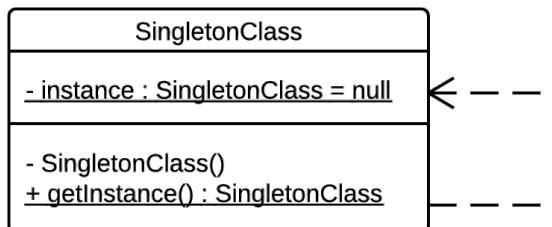


Figura 9: Diagramma ad alto livello del pattern Singleton

#### 4.8.3 Componenti che lo implementano

##### FAÇADE DEL SERVER

Il pattern Singleton pone un limite superiore stretto al numero di istanze che possono esistere di una determinata classe e perciò è utile utilizzarlo per poter controllare il numero di oggetti `server.StandardServerFacade` che in questo caso è pari a uno. L'unicità dell'oggetto façade garantisce la presenza di un solo punto di accesso alle funzionalità dell sotto-architettura server.

##### FAÇADE DEL PRESENTER

Il pattern Singleton è altresì utile per controllare il numero di istanze attive della classe `clientpresenter.StandardPresenterFacade` per gli stessi motivi evidenziati in precedenza, ossia l'unicità del punto di accesso alle funzionalità della sotto-architettura clientpresenter.

##### GESTIONE CONNESSIONE

La classe `server.connection.StandardConnectionHandler` è implementata come Singleton in modo da centralizzare la responsabilità di creare nuove connessioni in risposta alle esigenze dei client.

##### FAÇADE DELLA VISTA

La classe `clientview.StandardViewFacade` è progettata in accordo con il design pattern Singleton in modo da assicurare che in ogni momento ve ne sia una sola istanza attiva, responsabile di ricevere le richieste che giungono dal presenter.

##### GESTIONE GUI

Anche la classe `clientview.gui.GuiHandler` è implementata come Singleton in quanto permette di centralizzare la gestione delle collaborazioni fra le varie istanze attive delle classi grafiche.

### 4.9 State

#### 4.9.1 Scopo

Permette ad un oggetto di cambiare il suo comportamento al variare del suo stato interno, quindi a run-time. L'oggetto si comporterà come se avesse cambiato la sua classe.

#### 4.9.2 Diagramma esemplificativo

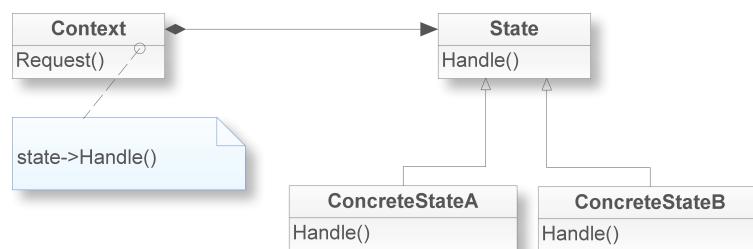


Figura 10: Diagramma ad alto livello del pattern State

#### 4.9.3 Componenti che lo implementano

##### GESTIONE DELLO STATO

Il pattern State permette di gestire gli utenti del sistema determinando un comportamento diverso per questi ultimi a seconda del loro stato.

È stata definita un'apposita gerarchia di stati che permette quindi di specializzare nella maniera più adatta alle necessità del sistema le operazioni sugli utenti senza bisogno di condizionali annidati.

## 5 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, e la rubrica dei contatti);
- il sistema proposto dal team è dotato di una parte server ed 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 clientpresenter (org.softwaresynthesis.mytalk.clientpresenter);
- il clientview (org.softwaresynthesis.mytalk.clientview).

Le specifiche di ogni sotto-architettura saranno definite in seguito, nelle relative sezioni.

Inoltre si fa presente che l'architettura generale, intesa come agglomerato delle tre sotto-architetture precedentemente elencate, fa uso del pattern MVP.

Sotto tale ottica la sotto-architettura server ricopre il ruolo di model, il 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 ad ogni sotto-architettura un ruolo specifico, si garantisce un alto livello di riutilizzo del codice (e.g. clientview comunica con il presenter poiché non conosce la logica di business del sistema).

Quindi in un futuro di potrebbe riprendere la vista oggi definita, e riutilizzarla in un altro progetto, andando solo a ridefinire, se necessario, un presente che riproponga una nuovo adattamento della parte logica.

Di seguito verranno proposte le sotto-architetture evidenziate. Di ogni una sarà dato un elenco dettagliato dei componenti che lo interessano. Si sottolinea che per componenti non si intende i sottopackage, ma gli agglomerati di classi (potenzialmente prese da package diversi) che concorrono ad un fine comune: la definizione delle funzionalità del componente trattato.

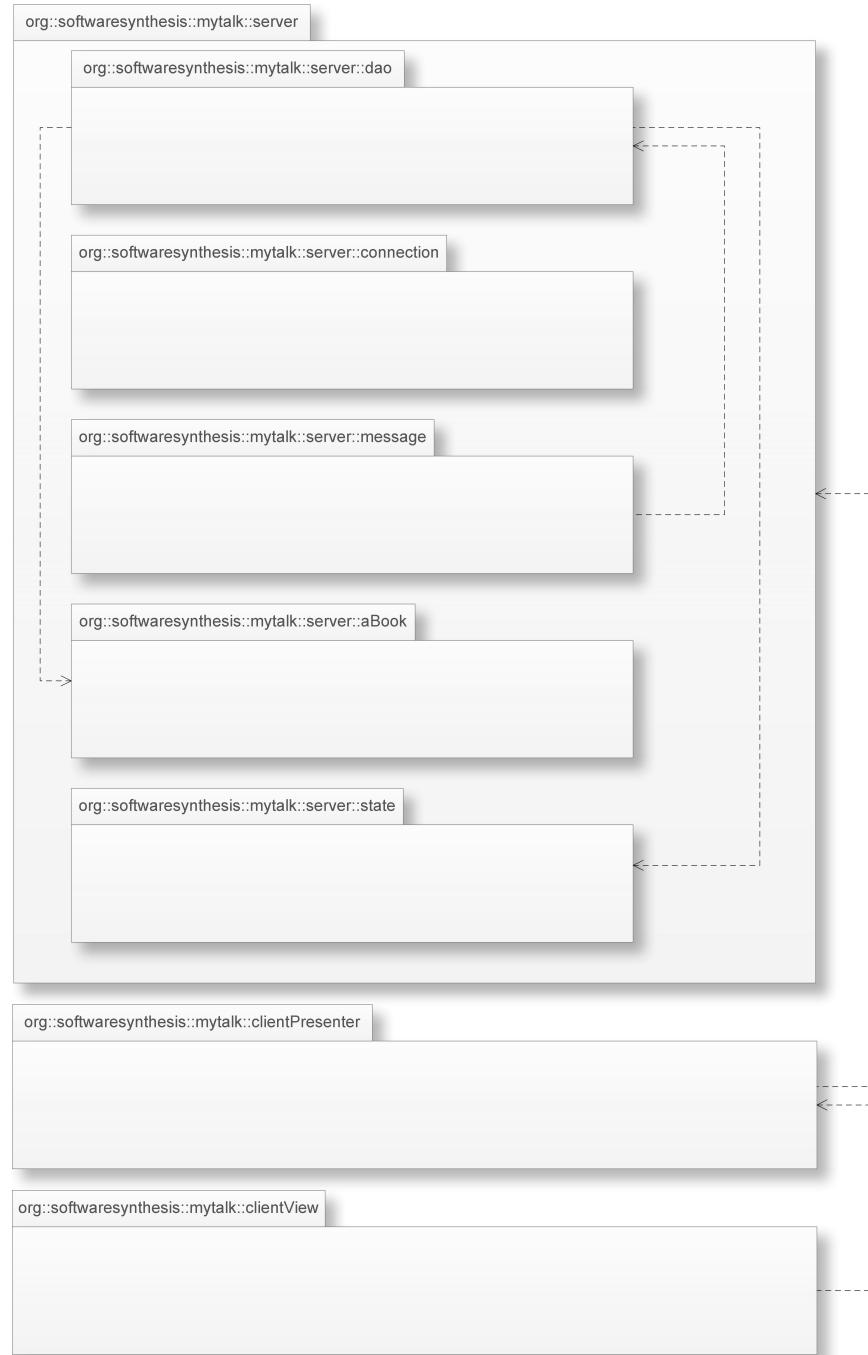


Figura 11: Diagramma dei Package - Generale

## 6 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.

### 6.1 Progettazione concettuale

Con la progettazione concettuale definiremo una struttura “concettuale” della base di dati. Tale struttura non rappresenta quella finale pronta per la creazione su DBMS, ma bensì un modello in grado di rappresentare il problema, svincolato da come esso debba essere logicamente rappresentato.

#### 6.1.1 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	PrimaryKey
Password	Password	NO	PrimaryKey
Domanda	Text	NO	
Risposta	Text	NO	
Nome	Varchar(100)	SI	
Cognome	Varchar(100)	SI	
Immagine	Varchar(150)	SI	

**CALLLIST:** è l'entità le cui istanze rappresentano lo storico delle chiamate effettuate da un utente (identificato nella tabella come “Utente\_chiamante”). UserData è caratterizzata dai seguenti attributi:

Nome attributo	Tipo	Opzionale	Vincoli
ID	int	NO	PrimaryKey
ID_chiamata	int	NO	
Utente_chiamante	Varchar(100)	NO	
Utente_ricevente	Varchar(100)	NO	
Data	Datetime	NO	

**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
ID_gruppo	Varchar(100)	NO	PrimaryKey
Nome	Varchar(100)	NO	
Proprietario	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
ID_messaggio	int	NO	PrimaryKey
Mittente	Varchar(100)	NO	
Destinatario	Varchar(100)	NO	

**AUDIOMESSAGES:** è un entità che specializza Messages e rappresenta gli audio messaggi lasciati nella segreteria di un utente. Gli attributi che la compongono sono:

Nome attributo	Tipo	Opzionale	Vincoli
Path	Varchar(150)	NO	PrimaryKey

**AUDIOVIDEOMESSAGES:** è un entità che specializza Messages e rappresenta i messaggi lasciati nella segreteria di un utente, per i quali si ha sia una traccia video che una traccia audio. Gli attributi che la compongono sono:

Nome attributo	Tipo	Opzionale	Vincoli
Path	Varchar(150)	NO	PrimaryKey

### 6.1.2 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. Si osservi che gli attributi contenuti in queste entità sono gli stessi. Infatti la necessità di mostrare la separazione logica deriva dalla volontà del team di evidenziare la separazione logica tra i due oggetti, motivata dalla possibilità di gestire le istanze delle due entità in modo diverso.

### 6.1.3 Associazione tra classi

**USERDATA - CALLLIST:** tale associazione rappresenta il legame che intercorre tra gli utenti e la CallList. Tale associazione è del tipo molti a uno da CallList a UserData, con totalità parziale verso CallList e totalità totale verso UserData.

**USERDATA - GROUPS (MOLTI A MOLTI):** tale associazione rappresenta il legame che intercorre tra gli utenti ed i gruppi, per la quale si è deciso di usare un associazione molti a molti con totalità parziale da entrambi i lati. Ciò significa che: un utente può comparire in uno o più gruppi (in ragione del fatto che potrebbe esserci qualcuno che registra l'utente A nella propria rubrica), ed un utente può possedere più gruppi.

**USERDATA - GROUPS (UNO A MOLTI):** tale associazione rappresenta il legame che intercorre tra il proprietario del gruppo e il gruppo stesso. L'associazione è del tipo uno a molti, a rappresentare come un gruppo sia visualizzabile unicamente dal proprietario, mentre un utente può avere (e di conseguenza visualizzare) più gruppi.

**USERDATA - MESSAGES:** tale associazione rappresenta il legame che intercorre tra gli utenti ed i messaggi registrati nella sua segreteria. L'associazione è 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 “visionabile” da un unico utente, il destinatario del messaggio.

### 6.1.4 Diagramma delle classi

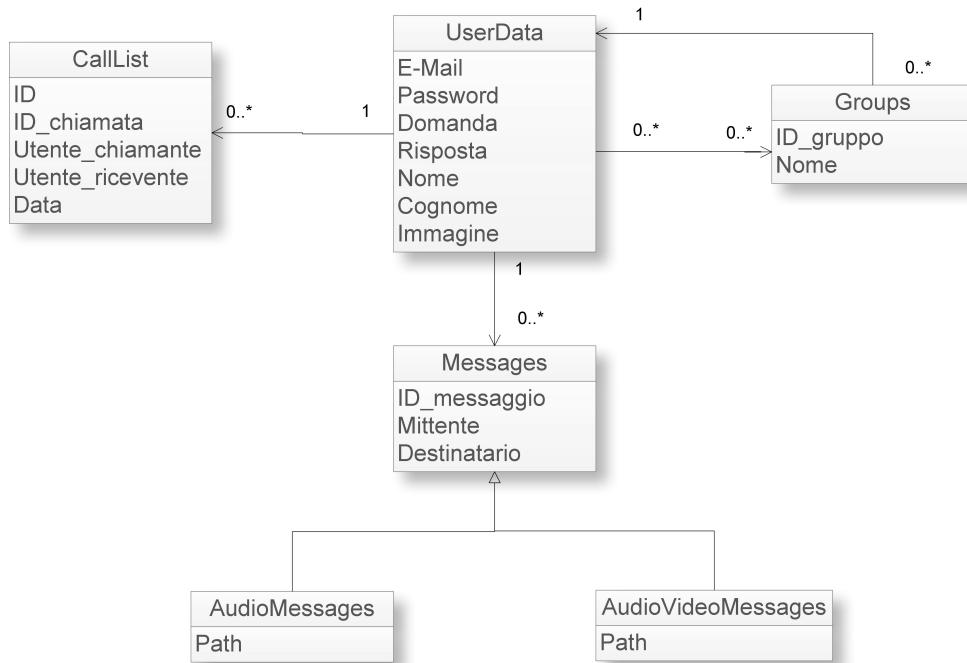


Figura 12: Diagramma delle classi - Schema concettuale database

## 6.2 Progettazione logica

Con la progettazione logica intendiamo costruire la struttura definitiva del database sulla base della progettazione concettuale. Quindi si deciderà come trasformare le gerarchie e in seguito le associazioni, al fine di restituire una struttura chiara e pronta per la creazione su DBMS. La lista delle classi proposta infine, e lo schema associato, rappresentano tale struttura.

### 6.2.1 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 un partizionamento orizzontale. Delle 3 entità iniziali sono rimaste solo AudioMessages e AudioVideoMessages, le quali incorporano nei loro attributi, gli attributi di Messages.

### 6.2.2 Rappresentazione delle associazioni

**USERDATA - CALLLIST:** tale associazione si è risolta trasformando: il campo ID\_chiamante in un ForeignKey verso UserData (per collegare la chiamata a colui che l'ha avviata), e analogamente usando il campo ID\_ricevente come Foreignkey verso UserData (per collegare la chiamata a chi l'ha ricevuta).

**USERDATA - GROUPS (MOLTI A MOLTI):** tale associazione si è risolta con la creazione di una terza tabella denominata AddressBook, che contiene una chiave esterna verso UserData e una chiave esterna verso Groups.

**USERDATA - GROUPS** (UNO A MOLTI: tale associazione si è risolta trasformando l'attributo “Proprietario” in una ForeignKey verso UserData.

**USERDATA - AUDIOMESSAGES**: tale associazione si è risolta inserendo una chiave esterna verso UserData, all'interno di AudioMessages.

**USERDATA - AUDIOVIDEOMESSAGES**: tale associazione si è risolta inserendo una chiave esterna verso UserData, all'interno di AudioVideoMessage.

### 6.2.3 Lista delle classi

UserData

Nome attributo	Tipo	Opzionale	Vincoli
E-Mail	Varchar(100)	NO	PrimaryKey
Password	Password	NO	PrimaryKey
Domanda	Text	NO	
Risposta	Text	NO	
Nome	Varchar(100)	SI	
Cognome	Varchar(100)	SI	
Immagine	Varchar(150)	SI	

CallList

Nome attributo	Tipo	Opzionale	Vincoli
ID	int	NO	PrimaryKey
ID_chiamata	int	NO	
Utente_chiamante	Varchar(100)	NO	ForeignKey verso UserData
Utente_ricevente	Varchar(100)	NO	ForeignKey verso UserData
Data	Datetime	NO	

AddressBook

Nome attributo	Tipo	Opzionale	Vincoli
ID_utente	Varchar(100)	NO	PrimaryKey e ForeignKey verso UserData
ID_gruppo	Varchar(100)	NO	PrimaryKey e ForeignKey verso Group

Groups

Nome attributo	Tipo	Opzionale	Vincoli
ID_gruppo	Varchar(100)	NO	PrimaryKey
Nome	Varchar(100)	NO	
Proprietario	Varchar(100)	NO	ForeignKey verso UserData

AudioMessages

Nome attributo	Tipo	Opzionale	Vincoli
Mittente	Varchar(100)	NO	
Destinatario	Varchar(100)	NO	ForeignKey verso UserData
Path	Varchar(150)	NO	PrimaryKey

## AudioVideoMessages

Nome attributo	Tipo	Opzionale	Vincoli
Mittente	Varchar(100)	NO	
Destinatario	Varchar(100)	NO	ForeignKey verso UserData
Path	Varchar(150)	NO	PrimaryKey

## 6.2.4 Diagramma delle classi

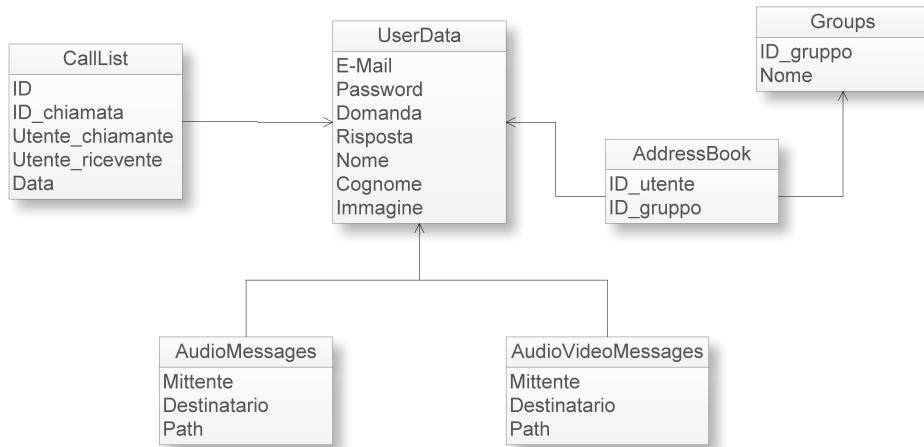


Figura 13: Diagramma delle classi - Schema logico Database

## 7 Architettura mytalk.server

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

- Gestione Database.
- Gestione connessione.
- Gestione rubrica.
- Gestione stato.
- Gestione segreteria.
- Façade del server.

I componenti sopracitati verranno definiti di seguito. Si sottolinea sin da ora che il server è l'unico in grado di comunicare con il database su cui regge l'applicativo.

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

### 7.1 Componenti evidenziati

#### 7.1.1 Gestione database

##### DESCRIZIONE:

Gestione Database è il componente che si occupa di rappresentare la struttura del database relazionale su cui poggia l'applicativo. Le singole classi in esso definite rappresentano quindi le tabelle del database. In termini tecnici le classi interne al componente Gestione database implementa il design pattern DAO.

Tramite questo componente, il sistema potrà quindi effettuare operazione di lettura e scrittura di entità all'interno del database. Le classi che costituiscono il componente dovranno quindi essere dotate di:

- metodi get per restituire i singoli attributi dell'istanza;
- metodi set per garantire un corretto inserimento dei dati prima di registrare l'istanza nel database.

Si informa inoltre che le classi di tale componente dovranno interagire con il framework Hibernate, al fine di ottenere lo scopo precisato.

##### DIAGRAMMA DELLE CLASSI:

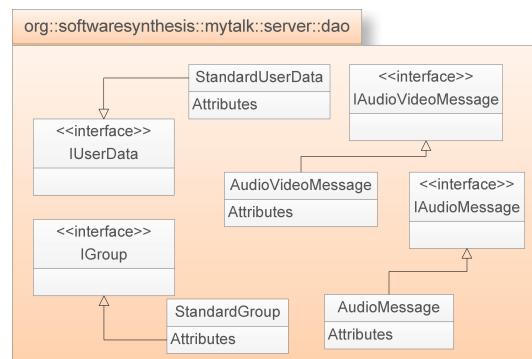


Figura 14: Diagramma delle classi - Gestione database

**CLASSI UTILIZZATE:**

- dao.AudioMessage
- dao.AudioVideoMessage
- dao.IAudioMessage
- dao.IAudioVideoMessage
- dao.IGroup
- dao.IUserData
- dao.StandardGroup
- dao.StandardUserData

**7.1.2 Gestione connessione****DESCRIZIONE:**

Tale componente ingloba le classi destinate a stabilire le routine di connessione. A tal fine è stata definita l'interfaccia di una classe Singleton `connection.ICommunicationHandler`, implementata da `connection.StandardCommunicationHandler` che ha il compito di creare oggetti connessione.

Le specifiche di tali oggetti sono descritte dall'interfaccia `connection.IConnection`, con la relativa implementazione `connection.WebRTCInfo`.

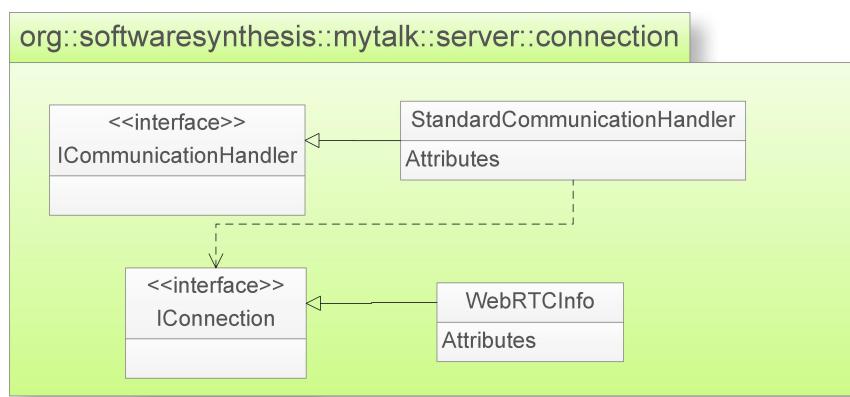
**DIAGRAMMA DELLE CLASSI:**

Figura 15: Diagramma delle classi - Gestione connessione

**CLASSI UTILIZZATE:**

- connection.ICommunicationHandler
- connection.StandardCommunicationHandler
- connection.IConnection
- connection.WebRTCInfo

**7.1.3 Gestione rubrica****DESCRIZIONE:**

La rubrica è organizzata in gruppi e sono previste due categorie di default: la *blacklist* e la *whitelist*.

L'utente può aggiungere ulteriori gruppi in base alle sue esigenze ma esclusivamente all'interno della *whitelist*. Per trattare in maniera omogenea i gruppi di contatti e i singoli contatti si è utilizzato il design pattern Composite.

In particolare, `abook.IContact` rappresenta l'interfaccia principale comune a ogni tipologia di contatto e viene estesa dalle due interfacce `dao.IUserData` e `dao.IGroup`.

Il componente comprende anche l'interfaccia `abook.IAddressBook` e la relativa implementazione `abook.AddressBook`. Nell'implementazione specificata `abook.Addressbook` vincola il sistema a garantire che ogni utente abbia i due gruppi di default sopra descritti, come richiesto dai requisiti.

Infine, la classe `abook(userDataProxy)` viene utilizzata come proxy per lo scambio di dati fra la sotto-architettura server e la sotto-architettura clientpresenter.

#### DIAGRAMMA DELLE CLASSI:

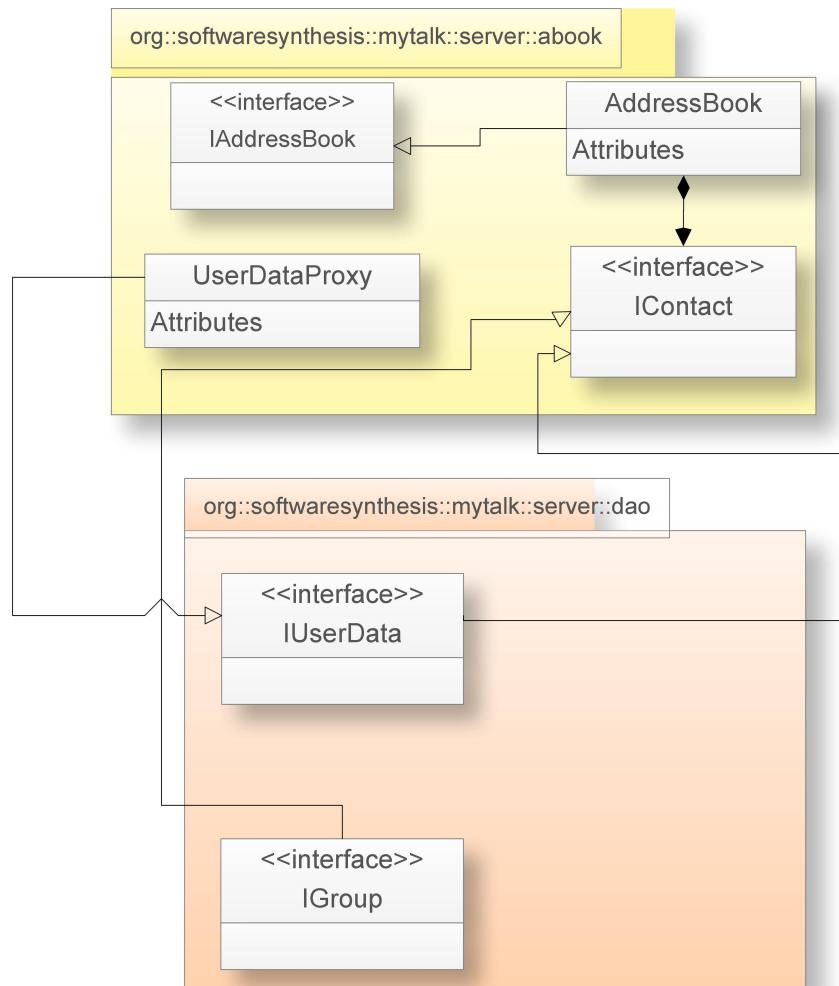


Figura 16: Diagramma delle classi - Gestione rubrica

#### CLASSI UTILIZZATE:

- `abook.AddressBook`
- `abook.IAddressBook`
- `abook.IContact`

- abook.UserDataProxy
- dao.IGroup
- dao.IUserData

#### 7.1.4 Gestione stato

##### DESCRIZIONE:

Le classi di tale componente sono utilizzate per gestire lo stato degli utenti, permettendo un comportamento diverso delle istanze di `dao.StandardUserData` a seconda dello stato in cui si trova l'utente corrispondente. Gli stati possibili sono in prima istanza “online” e “offline”, rappresentati dalle classi `state.StateOnline` e `state.StateOffline` rispettivamente.

Gli utenti che si trovano nello stato online possono trovarsi in due situazioni: “occupato” o “disponibile”, rappresentati a loro volta dalle classi `state.StateOccupied` e `state.StateAvailable`.

Se l'utente è impegnato in una conversazione con uno o più utenti, allora lo stato in cui si trova è “occupato”. Tuttavia, un utente può anche impostare manualmente il proprio stato ad “occupato” anche per segnalare di non essere disponibile a ricevere chiamate in ingresso.

La chiamata viene inoltre trattata in modo differente a seconda che l'utente si trovi nello stato “disponibile” o “occupato”/“offline”, dal momento che nel primo caso la chiamata va a buon fine mentre nel secondo verrà attivato il meccanismo di segreteria telefonica.

Inoltre, i cambiamenti di stato vengono notificati a tutti gli utenti presenti in rubrica tali che si trovano nello stato online.

##### DIAGRAMMA DELLE CLASSI:

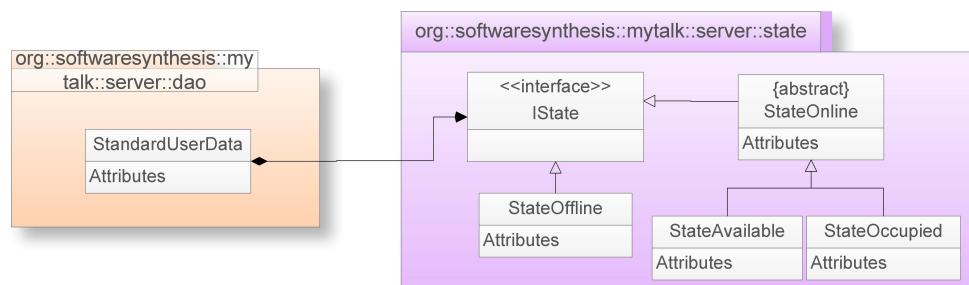


Figura 17: Diagramma delle classi - Gestione stato

##### CLASSI UTILIZZATE:

- `dao.StandardUserData`
- `state.IState`
- `state.StateAvailable`
- `state.StateOccupied`
- `state.StateOffline`
- `state.StateOnline`

### 7.1.5 Gestione segreteria

#### DESCRIZIONE:

Il sistema di segreteria telefonica corrisponde all'interfaccia `message. IMessageBox` e alla relativa implementazione `message.StandardMessageBox` che permettono un accesso centralizzato all'insieme di messaggi che un determinato utente ha ricevuto.

I messaggi audio e audio/video sul `server` sono rappresentati dalle classi `dao.IAudioMessage` (implementata da `dao.AudioMessage`) e `dao.IAudioVideoMessage` (implementata da `dao.AudioVideoMessage`) rispettivamente.

L'onere di caricare in memoria e gestire l'interno contenuto del messaggio è posticipato al momento di effettiva necessità mediante l'utilizzo dei *virtual proxy* corrispondenti alle classi `message.AudioMessageProxy` e `message.AudioVideoMessageProxy`.

#### DIAGRAMMA DELLE CLASSI:

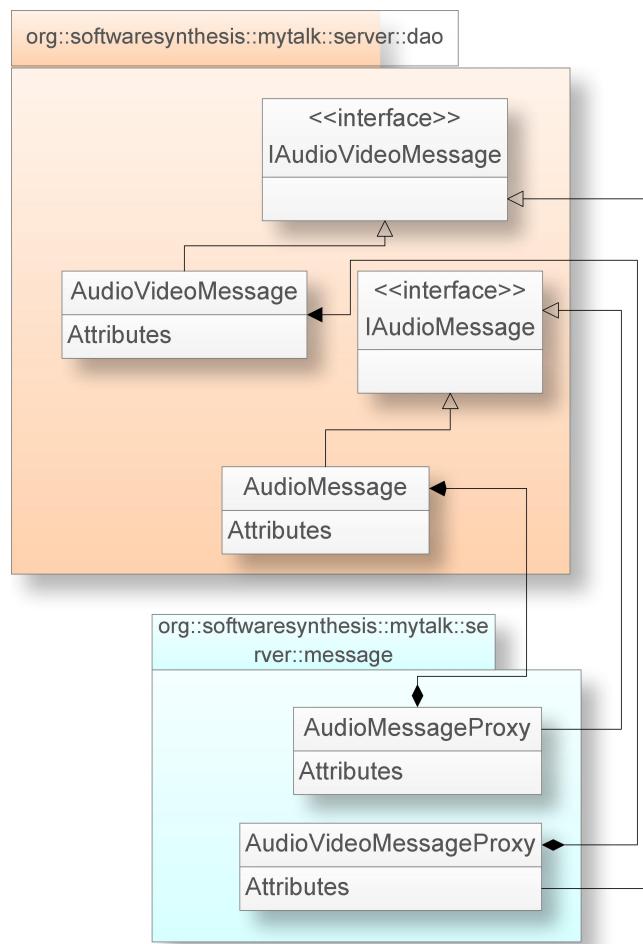


Figura 18: Diagramma delle classi - Gestione segreteria

#### CLASSI UTILIZZATE:

- `dao.AudioMessage`
- `dao.AudioVideoMessage`

- dao.IAudioMessage
- dao.IAudioVideoMessage
- message.AudioMessageProxy
- message.AudioVideoMessageProxy

#### 7.1.6 Façade del server

##### DESCRIZIONE:

L'interfaccia `I ServerFacade` e la relativa implementazione `StandardServerFacade`, nella quale si è scelto di applicare il design pattern Singleton, forniscono una sorta di interfaccia alle funzionalità offerte dalla sotto-architettura server ai componenti che risiedono nel client.

Le funzionalità esposte consentono di gestire i messaggi presenti in segreteria, le richieste di comunicazione con altri utenti il login/registrazione degli utenti.

##### DIAGRAMMA DELLE CLASSI:

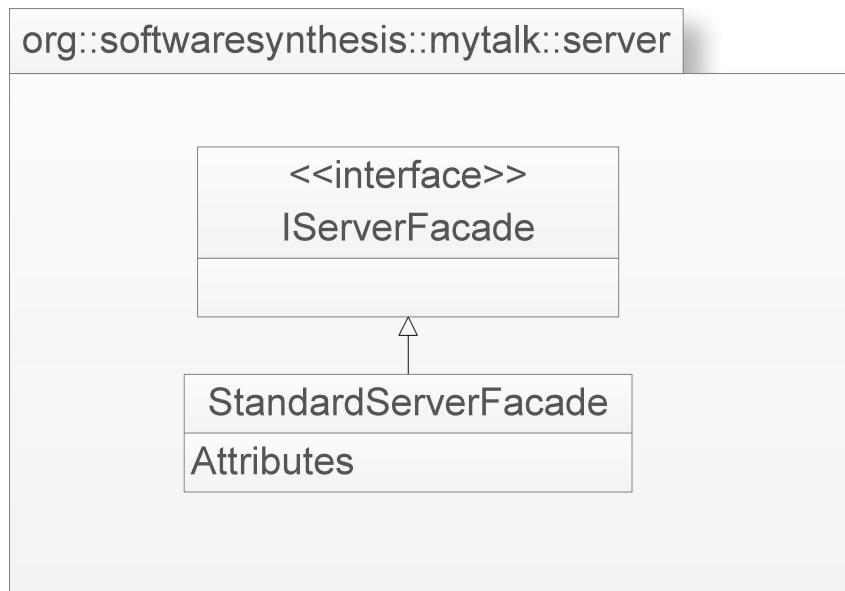


Figura 19: Diagramma delle classi - Façade del server

##### CLASSI UTILIZZATE:

- `I ServerFacade`
- `StandardServerFacade`

## 7.2 Diagramma delle classi

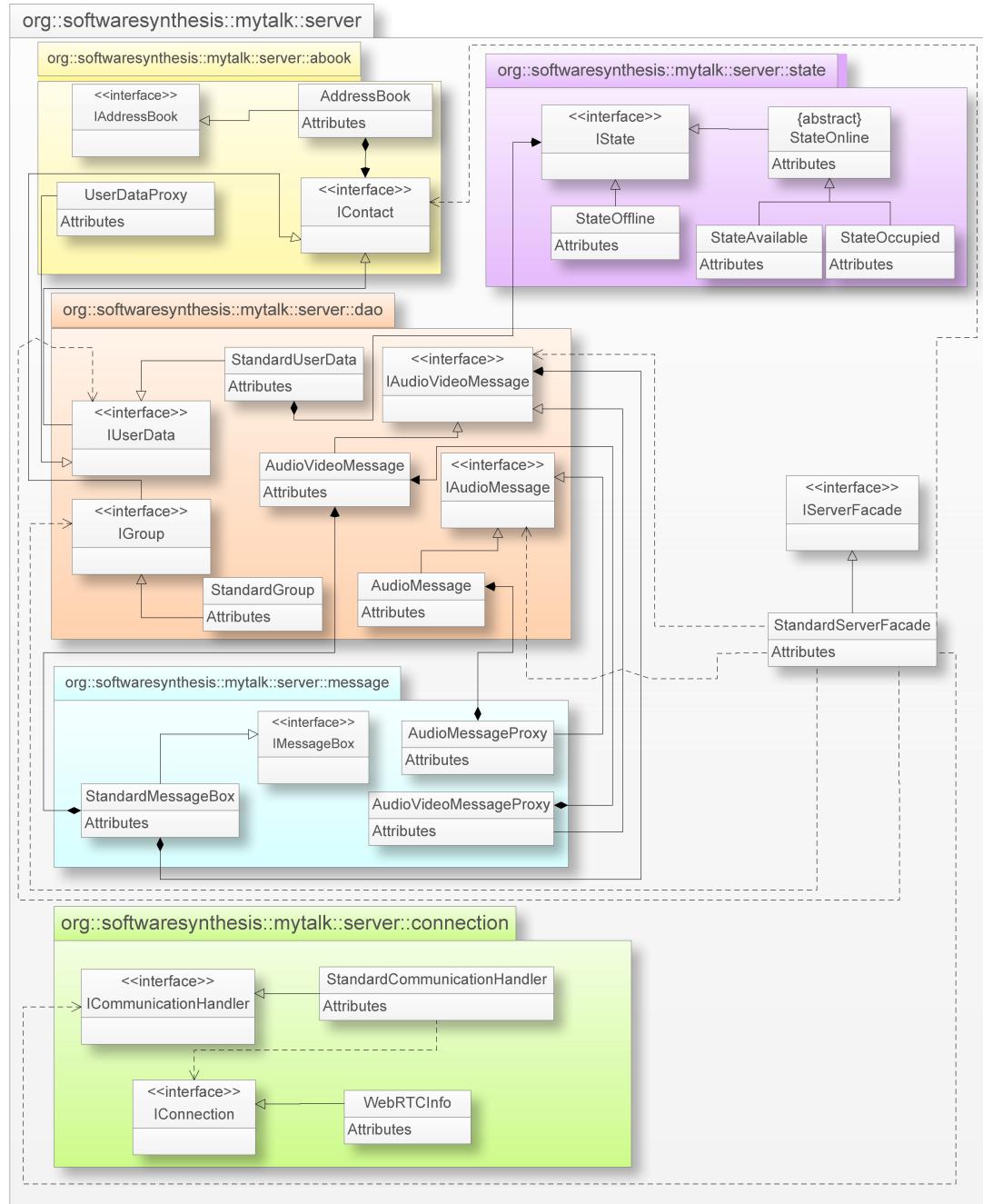


Figura 20: Diagramma delle classi - Architettura mytalk.server

## 8 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 usata da quest'ultimo.

Tale considerazione è conforme a l'utilizzo del design pattern MVP, già citato nella sezione "Design pattern", e alla quale rimandiamo per le specifiche tecniche del medesimo.

La sotto-architettura clientpresenter è per l'appunto la parte "presenter" dell'applicativo. I componenti che lo costituiscono devono quindi garantire una logica stabile di comunicazione con il server. Essi dovranno anche stabilire le procedure per interrogare il server (attraverso il suo façade) al fine di ottenere:

- i vari dati contenuti nel database;
- le "informazioni" necessarie per stabilire una connessione client-client.

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

- Gestione comunicazione;
- Façade del presenter.

Si rimanda alla sottosezione successiva per maggiori dettagli.

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

### 8.1 Logica di rete

La rete che viene a crearsi sotto l'architettura di MyTalk è suddivisa in "comunicazioni", ovvero gruppi di due o più utenti che comunicano tra di loro. Ogni comunicazione è formata dalle singole "connessioni" tra i vari client.

Questo capitolo serve a spiegare quale è la logica che governa la rete. Iniziamo definendo i tipi di client che la popolano.

#### NODO

Il Nodo è rappresentato da un semplice client che si trova all'interno di una comunicazione e che ha una unica connessione attiva verso un altro client.

#### SUPERNODO

Il Supernodo è un client con funzione di server. È usato nelle comunicazioni tra tre o più utenti per smistare le connessioni di tutti facendosi carico del traffico dei client.

Vediamo nel dettaglio come avviene una comunicazione. Essa nasce sempre tra due utenti, nel momento in cui uno desidera iniziare effettua una richiesta verso il server che la inoltra al destinatario. La comunicazione nasce così come un'unica connessione tra due client in modalità peer-to-peer dove entrambi gli utenti hanno il ruolo di Nodo. Una comunicazione potrà iniziare solo attraverso tale metodologia, in quanto non vengono aperte comunicazioni iniziali tra più di due utenti.

Da questa situazione uno dei due utenti può richiederne l'aggiunta di un ulteriore soggetto. Dopo questa richiesta, il Nodo che la ha eseguita diventa un Supernodo con attive sia la connessione con il vecchio Nodo sia quella con il nuovo Nodo appena aggiunto.

Questa operazione crea una connessione fra tre Nodi estendibile dal Supernodo e dagli altri Nodi attraverso una richiesta al Supernodo stesso.

Ogni Nodo può togliersi dalla comunicazione chiudendo la connessione con il Supernodo. Se è il Supernodo stesso a volersi togliere dalla comunicazione, l'intera comunicazione cade e tutti gli utenti vengono disconnessi.

### 8.1.1 L'idea iniziale

La logica di gestione della rete scelta dal team può rendere poco stabile le comunicazioni. Tale problematica è stata analizzata a lungo, e al fine di rendere le comunicazioni più stabili e solide era stata presa in considerazione un ulteriore soluzione alternativa sempre basata sul modello concettuale di Nodi e Supernodi, tuttavia è stata scartata dopo una breve analisi preliminare.

La differenza principale di tale soluzione rispetto a quella adottata è nella gestione dinamica del Supernodo, che viene scelto in base alla sua capacità di banda e alla possibilità di mantenere attive, con la massima qualità, tutte le connessioni con i Nodi della comunicazione.

Questa soluzione avrebbe richiesto un test di qualità della rete per ogni Nodo verso il nuovo possibile Supernodo, operazione di difficile progettazione e con complessità elevata, pari a  $\omega(n - 1)$  con n Nodi attivi nella comunicazione.

Al fine di garantire sempre la migliore qualità di connessione era stata presa inoltre in considerazione l'idea di controllare periodicamente la qualità del Supernodo e degli altri Nodi in modo da sostituire il Supernodo se necessario. Purtroppo anche questa soluzione richiederebbe dei calcoli di complessità  $\omega(n * n - 1)$  con n Nodi attivi nella comunicazione.

Con una corretta implementazione questa soluzione avrebbe permesso una migliore qualità del prodotto MyTalk, il gruppo ha tuttavia optato per la prima implementazione presentata in quanto le risorse disponibili non erano sufficienti per terminare la seconda.

## 8.2 Componenti evidenziati

### 8.2.1 Gestione comunicazione

#### DESCRIZIONE:

È il componente che definisce i client con cui sta comunicando o con cui desidera comunicare l'utente. Tale componente è costituito da un'interfaccia clientpresenter.IClient che rappresenta un modello di client con cui interagire (possibilità di stabilire una connessione, possibilità di estendere una comunicazione e possibilità di verificare se il client è Supernodo).

In pratica nel momento in cui l'utente A desidera stabilire una comunicazione con l'utente B, questi si farà restituire dal server le informazioni necessarie. Con tali informazioni, quindi, sarà creata un'istanza di clientpresenter.StandardClient (implementazione dell'interfaccia clientpresenter.IClient), dalla quale potrà essere abilitata la comunicazione.

#### DIAGRAMMA DELLE CLASSI:

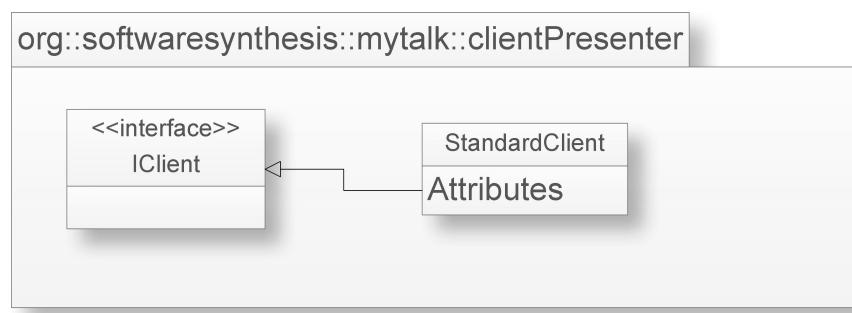


Figura 21: Diagramma delle classi - Gestione comunicazione

#### CLASSI UTILIZZATE:

- IClient
- StandardClient

### 8.2.2 Façade del presenter

#### DESCRIZIONE:

Rappresenta l'interfaccia d'accesso verso la sotto-architettura clientpresenter. Tale componente svincola l'interfaccia grafica dal dover conoscere l'esatta sequenza di chiamata dei metodi per portare a conseguimento una determinata procedura.

I metodi del componente Façade del presenter hanno la cognizione dell'iter di chiamate a metodo per portare a termine correttamente la procedura. Il componente in esame richiede la definizione di un'interfaccia e una sua implementazione standard. Rimandiamo a tali classi per un approfondimento sul comportamento da seguire.

#### DIAGRAMMA DELLE CLASSI:

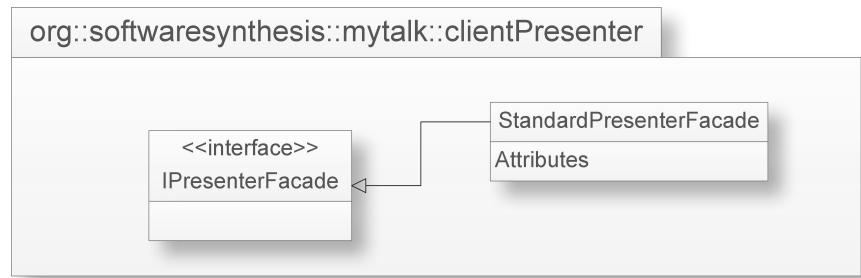


Figura 22: Diagramma delle classi - Façade del presenter

#### CLASSI UTILIZZATE:

- IPresenterFacade
- StandardPresenterFacade

### 8.3 Diagramma delle classi

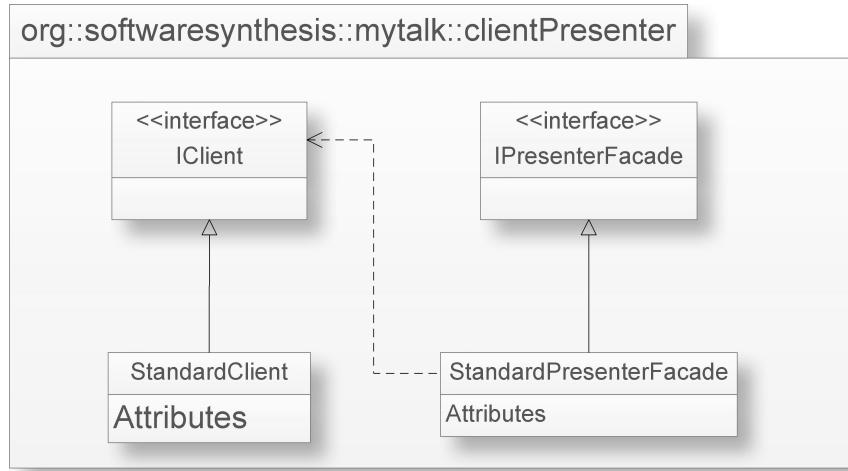


Figura 23: Diagramma delle classi - Architettura mytalk.clientpresenter

## 9 Architettura mytalk.clientview

Come già accennato in precedenza, nella sezione introduttiva all'architettura e nella sezione del clientpresenter, clientview ha il compito di definire la logica 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 programmatore 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 iterazione, da parte dell'utente finale, con l'applicativo MyTalk.

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

Per quanto riguarda la componentistica, la sotto-architettura clientview è dotata di:

- Façade della vista
- Gestione GUI

Per le specifiche dei componenti sopra citati, si rimanda alla sezione successiva.

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

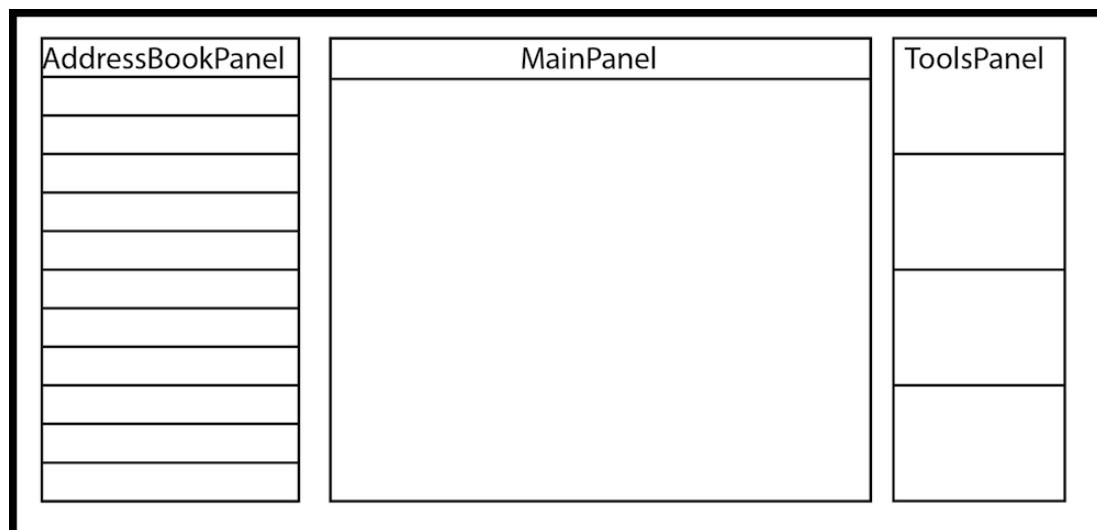


Figura 24: Rappresentazione ad alto livello della GUI

In figura 24 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.

In particolare, 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, varia 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.

## 9.1 Componenti evidenziati

### 9.1.1 Façade della vista

#### DESCRIZIONE:

Il compito di questo componente è rappresentare l’intera sotto-architettura clientview, per come viene vista dalle classi del presenter. In tal modo è possibile rendere indipendenti le classi concrete della sotto-architettura clientview in quanto le dipendenze funzionali provenienti dall’esterno sono accentrate nel Façade.

In particolare, tramite le operazioni che sono dichiarate nell’interfaccia `IViewFacade` si rende possibile l’aggiornamento dinamico a tempo di esecuzione della vista e, di conseguenza, di quello che viene visualizzato dall’utente finale del sistema.

#### DIAGRAMMA DELLE CLASSI:

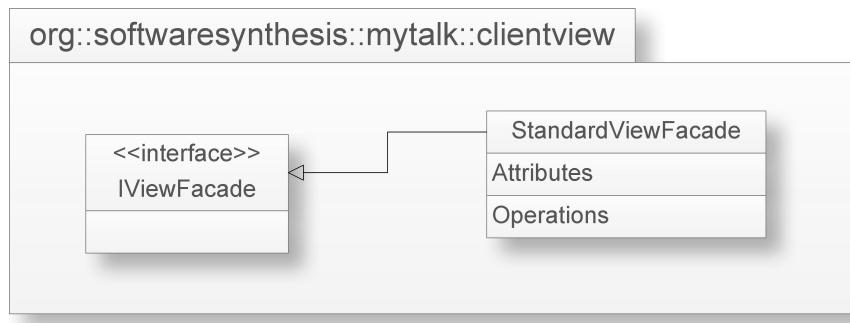


Figura 25: Diagramma delle classi - Façade della vista

#### CLASSI UTILIZZATE:

- `IViewFacade`
- `StandardViewFacade`

### 9.1.2 Gestione GUI

#### DESCRIZIONE:

Questo componente si occupa della gestione dell’interfaccia grafica presentata all’utente finale. Le classi in esso contenute servono pertanto per la rappresentazione delle funzionalità offerte dal sistema all’utente.

Si noti che questo componente demanda tutta la logica al presenter, tramite il Façade del presenter, e quindi è relativo solo ed esclusivamente alla grafica. Inoltre l’interazione con esso è centralizzata all’interno della classe `gui.GUIHandler`, ciò significa che tutte le altre classi del package `gui` non interagiscono direttamente con il presenter.

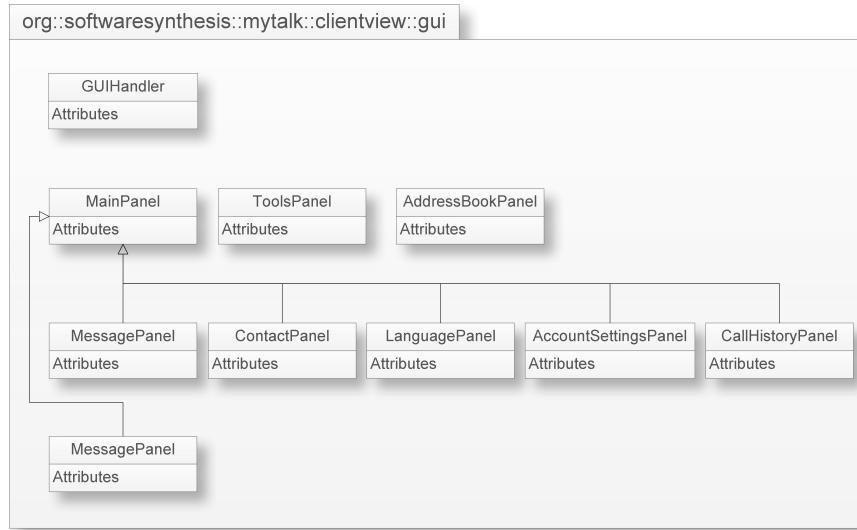
**DIAGRAMMA DELLE CLASSI:**

Figura 26: Diagramma delle classi - Gestione GUI

**CLASSI UTILIZZATE:**

- `gui.GUIHandler`
- `gui.MainPanel`
- `gui.ToolsPanel`
- `gui.AddressBookPanel`
- `gui.ContactPanel`
- `gui.SearchResultPanel`
- `gui.MessagePanel`
- `gui.LanguagePanel`
- `gui.AccountSettingsPanel`
- `gui.CallHistoryPanel`

## 9.2 Diagramma delle classi

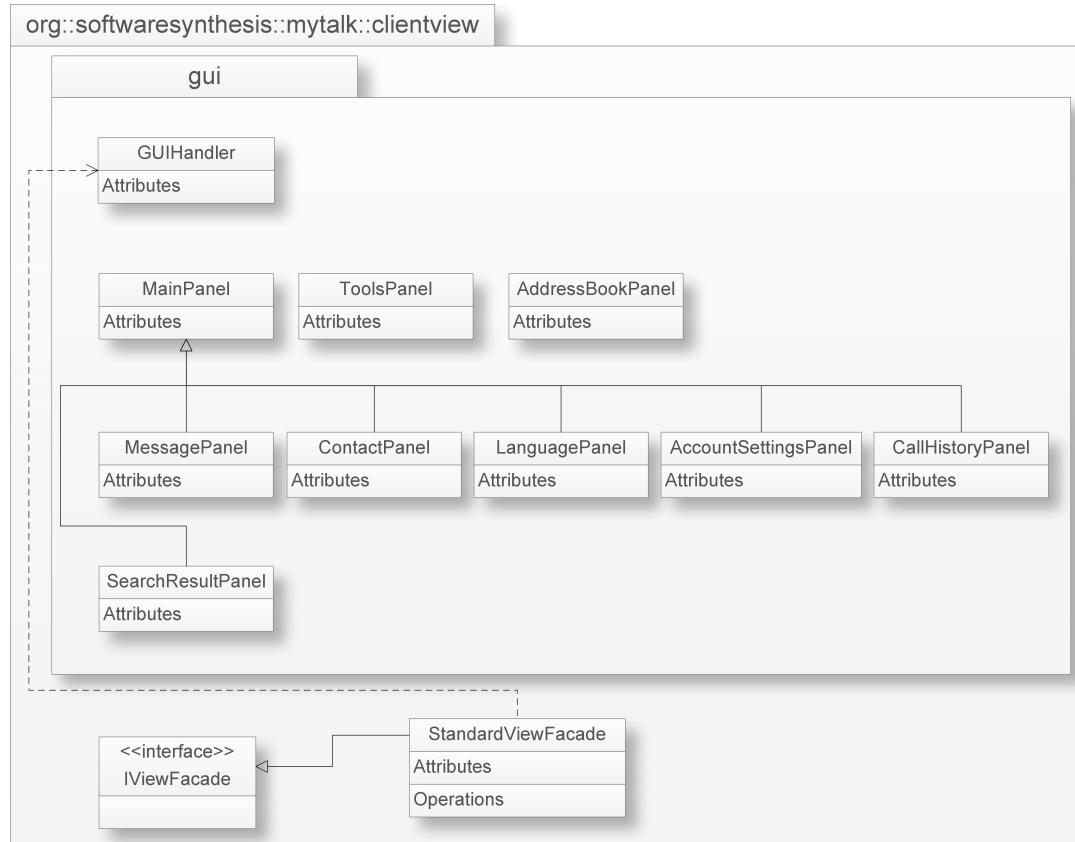


Figura 27: Diagramma delle classi - Architettura mytalk.clientview

## 10 Descrizione delle classi

### 10.1 Package org.softwaresynthesis.mytalk.server.dao

#### 10.1.1 IAudioMessage

**DESCRIZIONE:**

Interfaccia per i messaggi audio della segreteria telefonica che viene implementata dalle classi `AudioMessage` e dal relativo proxy `server.message.AudioMessageProxy`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione segreteria

#### 10.1.2 IAudioVideoMessage

**DESCRIZIONE:**

Interfaccia per i messaggi audio/video della segreteria telefonica, a sua volta implementata dalle classi `AudioVideoMessage` e dal suo proxy `server.message.AudioVideoMessageProxy`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione segreteria

#### 10.1.3 AudioMessage

**DESCRIZIONE:**

Classe che rappresenta un messaggio audio nella segreteria telefonica di un utente, implementa l'interfaccia `IAudioMessage`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione segreteria

#### 10.1.4 AudioVideoMessage

**DESCRIZIONE:**

Classe che rappresenta un messaggio audio/video nella segreteria telefonica di un utente, implementa l'interfaccia `IAudioVideoMessage`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione segreteria

#### 10.1.5 IGroup

**DESCRIZIONE:**

Interfaccia per i gruppi interni alla rubrica, prevede un'operazione astratta `add(IUserData)` per l'aggiunta di un nuovo contatto al gruppo e un'operazione `remove(IUserData)` per la sua rimozione.

Estende inoltre l'interfaccia `server.abook.IContact` comune a tutti i contatti della rubrica e richiede per l'applicazione del design pattern Composite.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione rubrica

### 10.1.6 StandardGroup

**DESCRIZIONE:**

Implementazione dell'interfaccia `IGroup`, ogni gruppo è dotato di un nome e raccoglie in sé zero o più istanze di classi sottotipo di `server.abook.IContact`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione database

### 10.1.7 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:**

- Gestione database
- Gestione rubrica
- Gestione connessione

### 10.1.8 StandardUserData

**DESCRIZIONE:**

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

**COMPONENTI CHE NE FANNO USO:**

- Gestione database
- Gestione stato

## 10.2 Package org.softwaresynthesis.mytalk.server.connection

### 10.2.1 ICommunicationHandler

**DESCRIZIONE:**

Interfaccia per la gestione delle richieste per ottenere le informazioni necessarie a un client per stabilire una comunicazione con altri client. In particolare le operazioni in essa dichiarate dovranno ritornare un'istanza di un oggetto che implementa `IConnection`.

**COMPONENTI CHE NE FANNO USO:**

- Gestione connessione

### 10.2.2 StandardCommunicationHandler

**DESCRIZIONE:**

Implementazione che si fornisce dell'interfaccia `ICommunicationHandler`. In tale classe, le informazioni ritornate al client sono informazioni necessarie a stabilire una connessione di tipo WebRTC (ritorna un'istanza di `WebRTCInfo`).

**COMPONENTI CHE NE FANNO USO:**

- Gestione connessione

### 10.2.3 IConnection

**DESCRIZIONE:**

Interfaccia mediante la quale è possibile recuperare i dati necessari a un client al fine di stabilire una connessione con un altro client.

**COMPONENTI CHE NE FANNO USO:**

- Gestione connessione
- Gestione comunicazione

### 10.2.4 WebRTCInfo

**DESCRIZIONE:**

Implementazione dell'interfaccia **IConnection** che rappresenta le informazioni usate da un client per stabilire una connessione WebRTC con un altro client.

**COMPONENTI CHE NE FANNO USO:**

- Gestione connessione

## 10.3 Package org.softwaresynthesis.mytalk.server.abook

### 10.3.1 IContact

**DESCRIZIONE:**

Interfaccia condivisa da tutti i contatti della rubrica di un utente, permette di trattare in modo uniforme gli utenti singoli e i gruppi secondo quanto previsto dal design pattern Composite.

Dichiara la presenza di operazioni astratte per recuperare l'accesso alle informazioni sottostanti (ad esempio il nome di un utente o di un gruppo).

**COMPONENTI CHE NE FANNO USO:**

- Gestione rubrica

### 10.3.2 IAddressBook

**DESCRIZIONE:**

Interfaccia che raccoglie le operazioni sulla rubrica e permette di controllare l'accesso alle strutture dati che la implementano.

**COMPONENTI CHE NE FANNO USO:**

- Gestione rubrica

### 10.3.3 UserDataProxy

**DESCRIZIONE:**

La classe implementa l'interfaccia **server.dao.IUserData** e rappresenta un proxy per i dati degli utenti che risiedono sul server.

Le istanze di questa classe vengono restituite da un opportuno Factory Method dichiarato nell'interfaccia **server.ServerFacade** e hanno il ruolo di rappresentare sui client degli oggetti che risiedono in un diverso spazio di indirizzamento (sul server) nonché di controllare gli accessi per esigenze di protezione.

**COMPONENTI CHE NE FANNO USO:**

- Gestione rubrica

#### 10.3.4 AddressBook

**DESCRIZIONE:**

Classe che implementa l'interfaccia `IAddressBook` e rappresenta quindi la rubrica di un determinato utente. Contiene un riferimento a `IContact` che corrisponde al nodo padre della struttura dati ad albero contenente la rubrica.

**COMPONENTI CHE NE FANNO USO:**

- Gestione rubrica

### 10.4 Package org.softwaresynthesis.mytalk.server.state

#### 10.4.1 IState

**DESCRIZIONE:**

Interfaccia padre della gerarchia di stati che gli utenti possono assumere nel corso dell'interazione con il sistema. La composizione delle istanze di `server.dao.StandardUserData` con oggetti sottotipo di `IState` cui inoltrano le richieste ricevute dall'esterno permette di cambiare dinamicamente il comportamento degli oggetti che rappresentano gli utenti.

**COMPONENTI CHE NE FANNO USO:**

- Gestione stato
- Gestione connessione

#### 10.4.2 StateOnline

**DESCRIZIONE:**

Classe astratta che implementa l'interfaccia `IState` e viene estesa da tutti gli stati che corrispondono alla presenza "online" dell'utente. È ulteriormente specializzata dalle sottoclassi concrete `StateAvailable` e `StateOccupied`, le quali hanno facoltà di determinare il reale comportamento degli utenti online.

**COMPONENTI CHE NE FANNO USO:**

- Gestione stato

#### 10.4.3 StateOffline

**DESCRIZIONE:**

Classe concreta della gerarchia degli stati che implementa l'interfaccia `IState` e corrisponde alla mancata presenza online dell'utente che la possiede. Un utente che si trova nello stato "offline" non può essere contattato direttamente ma solo attraverso la segreteria telefonica.

**COMPONENTI CHE NE FANNO USO:**

- Gestione stato

#### 10.4.4 StateAvailable

**DESCRIZIONE:**

Classe concreta della gerarchia degli stati che corrisponde alla presenza online di un utente e alla sua disponibilità ad accettare comunicazioni in ingresso.

**COMPONENTI CHE NE FANNO USO:**

- Gestione stato

#### 10.4.5 StateOccupied

**DESCRIZIONE:**

Classe concreta della gerarchia degli stati che corrisponde alla mancata disponibilità di un utente a ricevere ulteriori comunicazioni in ingresso dal momento che si trova impegnato in un'altra conversazione. Gli utenti che si trovano in questo stato non possono essere contattati direttamente ma solo attraverso la possibilità di lasciare un messaggio in segreteria.

**COMPONENTI CHE NE FANNO USO:**

- Gestione stato

### 10.5 Package org.softwaresynthesis.mytalk.server.message

#### 10.5.1 IMessagesBox

**DESCRIZIONE:**

Interfaccia che rappresenta, ad alto livello, la segreteria telefonica dell'utente cui è associata. Tramite opportune operazioni permette di accedere all'elenco dei messaggi audio o audio/video che l'utente ha ricevuto nel periodo in cui era in linea o non era disponibile perché impegnato in una conversazione.

**COMPONENTI CHE NE FANNO USO:**

- Gestione segreteria

#### 10.5.2 StandardMessageBox

**DESCRIZIONE:**

Classe concreta che implementa l'interfaccia `IMessagesBox` e rappresenta dunque la rubrica associata a un determinato utente.

**COMPONENTI CHE NE FANNO USO:**

- Gestione segreteria

#### 10.5.3 AudioMessageProxy

**DESCRIZIONE:**

Proxy virtuale ad uso dei client per i messaggi audio lasciati ad un utente in segreteria. La classe implementa l'interfaccia `server.dao.IAudioMessage` ogni sua istanza è costruita per composizione con un riferimento a `server.dao.AudioMessage`, cui inoltra le richieste dopo aver eventualmente scaricato il messaggio dal server.

**COMPONENTI CHE NE FANNO USO:**

- Gestione segreteria

#### 10.5.4 AudioVideoMessageProxy

**DESCRIZIONE:**

Proxy virtuale ad uso dei client per i messaggi audio/video in segreteria. La classe implementa l'interfaccia `server.dao.AudioVideoMessage` ed è costruita, similmente a `AudioMessageProxy` con un'istanza di `server.dao.AudioVideoMessage` per limitare il traffico di rete e ottimizzare il consumo di memoria del client.

**COMPONENTI CHE NE FANNO USO:**

- Gestione segreteria

## 10.6 Package org.softwaresynthesis.mytalk.server

### 10.6.1 IServerFacade

#### DESCRIZIONE:

Interfaccia che contiene tutte le operazioni richieste dai componenti della sotto-architettura clientpresenter inerenti alla registrazione, all'autenticazione, alla gestione della rubrica e della segreteria nonché delle connessioni con altri utenti.

In quest'ultimo caso le operazioni richieste vengono a loro volta inoltrate al componente responsabile della gestione della connessione (`server.connection.IConnectionHandler`).

#### COMPONENTI CHE NE FANNO USO:

- Façade del server
- Façade del presenter
- Gestione comunicazione

### 10.6.2 StandardServerFacade

#### DESCRIZIONE:

Questa classe Singleton è l'implementazione dell'interfaccia `IServerFacade` e contiene, in particolare, i metodi che rendono concrete le operazioni corrispondenti ai Factory Method dichiarati nell'interfaccia citata (per la gestione dei messaggi in segreteria e degli utenti nella rubrica).

#### COMPONENTI CHE NE FANNO USO:

- Façade del server

## 10.7 Package org.softwaresynthesis.mytalk.clientpresenter

### 10.7.1 IClient

Interfaccia che raccoglie le operazioni astratte che è possibile invocare su un `client` astratto del sistema corrispondenti, ad esempio, alla ricezione di una chiamata in ingresso oppure all'interrogazione sullo stato (nodo semplice o Supernodo) di un determinato `client`.

#### DESCRIZIONE:

#### COMPONENTI CHE NE FANNO USO:

- Gestione comunicazione

### 10.7.2 StandardClient

#### DESCRIZIONE:

Implementazione dell'interfaccia `IClient` che incapsula al suo interno la logica di eventuale ritrasmissione dei dati in ingresso (nel caso in cui il `client` sia un Supernodo) e di invio dei dati prodotti in locale ai `client` con cui è stata instaurata una connessione.

#### COMPONENTI CHE NE FANNO USO:

- Gestione comunicazione

### 10.7.3 IPresenterFacade

#### DESCRIZIONE:

Interfaccia che raccoglie la dichiarazione di tutte le operazioni astratte con cui si suppone che la vista realizzi lo scambio di messaggi con il presenter. Tramite queste operazioni deve dunque essere possibile, ad esempio, gestire l'autenticazione di un utente, interrogare la segreteria telefonica, modificare lo stato e le informazioni personali e scaricare la rubrica.

#### COMPONENTI CHE NE FANNO USO:

- Façade del presenter
- Gestione GUI

### 10.7.4 StandardPresenterFacade

#### DESCRIZIONE:

Questa classe rappresenta un'applicazione del design pattern Singleton e implementa inoltre l'interfaccia IPresenterFacade. Fornisce pertanto dei metodi concreti utilizzati per la comunicazione fra la vista e il server, elaborando qualora necessario i dati ricevuti in input e inoltrando le richieste al componente Façade del server.

#### COMPONENTI CHE NE FANNO USO:

- Façade del presenter

## 10.8 Package org.softwaresynthesis.mytalk.clientview

### 10.8.1 IViewFacade

#### DESCRIZIONE

Interfaccia che dichiara le operazioni astratte che possono essere richiamate sulla vista a partire dai componenti della sotto-architettura clientpresenter.

#### COMPONENTI CHE NE FANNO USO:

- Façade della vista
- Façade del presenter

### 10.8.2 StandardViewFacade

#### DESCRIZIONE

Implementazione dell'interfaccia IViewFacade che contiene la definizione concreta dei metodi in essa dichiarati.

#### COMPONENTI CHE NE FANNO USO:

- Façade della vista

## 10.9 Package org.softwaresynthesis.clientview.gui

### 10.9.1 GUIHandler

#### DESCRIZIONE

L'istanza di questa classe ha il compito di inoltrare ai componenti delle diverse sotto-architetture le richieste provenienti dalle altre classi di questo package, in quanto unico possessore di un riferimento di tipo IServerFacade.

Inoltre, tale classe ha la responsabilità di controllare l'attivazione e gli aggiornamenti dinamici nel corso dell'esecuzione di tutti gli oggetti di classi grafiche istanziate.

**COMPONENTI CHE NE FANNO USO:**

- Gestione GUI

**10.9.2 AddressBookPanel****DESCRIZIONE**

Pannello che permette la visualizzazione della rubrica dell'utente connesso al sistema, e di accedere alle funzionalità di amministrazione della rubrica stessa (come l'aggiunta o la rimozione di utenti/gruppi, lo spostamento di un utente da un gruppo a un altro oppure l'ordinamento).

**COMPONENTI CHE NE FANNO USO:**

- Gestione GUI

**10.9.3 MainPanel****DESCRIZIONE**

Classe padre della gerarchia di oggetti grafici che possono comparire nella sezione principale dell'interfaccia utente. La parte comune a tutte le sottoclassi permette all'utente di impostare il proprio stato e di uscire dall'applicazione. Ogni sua specializzazione si rivolge a **GUIHandler** per inoltrare i comandi verso il componente Façade del presenter.

**COMPONENTI CHE NE FANNO USO:**

- Gestione GUI

**10.9.4 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.

**COMPONENTI CHE NE FANNO USO:**

- Gestione GUI

**10.9.5 SearchResultPanel****DESCRIZIONE**

Componente grafica che estende la classe **MainPanel** e visualizza i risultati della ricerca di un utente. La sua visualizzazione avviene in seguito all'uso della componente grafica di ricerca presente nel pannello degli strumenti.

- Gestione GUI

**10.9.6 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.

**COMPONENTI CHE NE FANNO USO:**

- Gestione GUI

### 10.9.7 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`.

#### COMPONENTI CHE NE FANNO USO:

- Gestione GUI

### 10.9.8 LanguagePanel

#### DESCRIZIONE

Questa sottoclasse di `MainPanel` è impiegata per permettere da parte dell'utente finale la selezione della lingua desiderata. La visualizzazione del pannello avviene in seguito all'attivazione del pulsante corrispondente situato nel pannello degli strumenti.

#### COMPONENTI CHE NE FANNO USO:

- Gestione GUI

### 10.9.9 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:

- Gestione GUI

### 10.9.10 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`.

#### COMPONENTI CHE NE FANNO USO:

- Gestione GUI

## 11 Conclusioni sull'architettura

### 11.1 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.

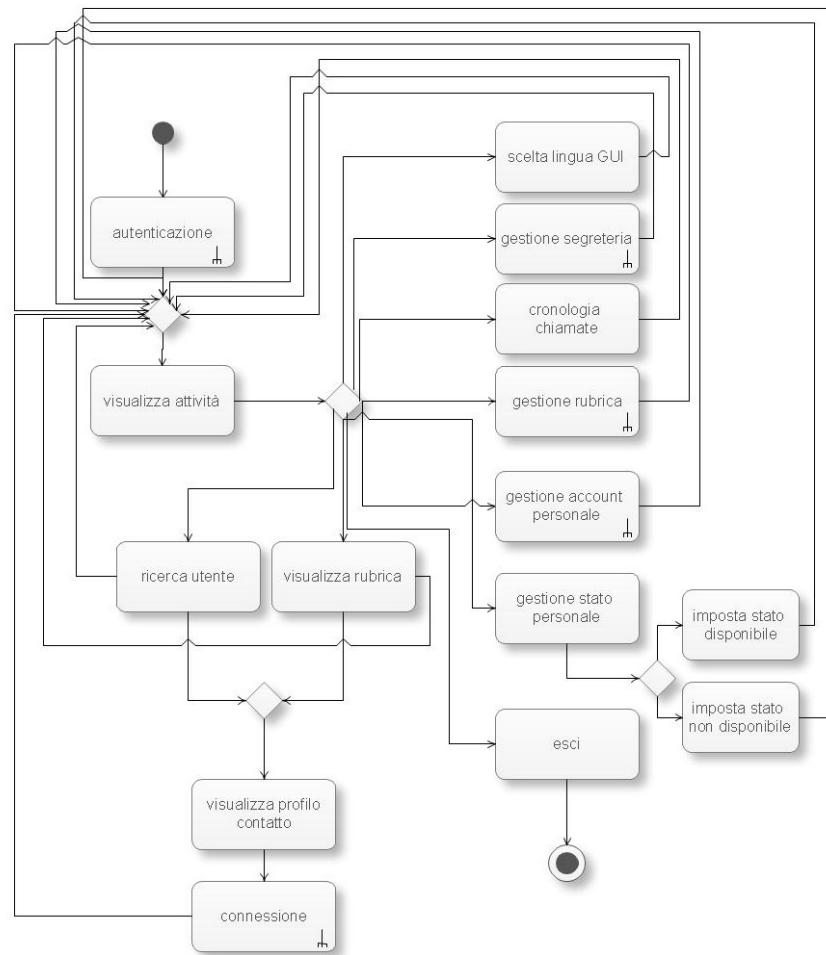


Figura 28: Diagramma di attività generale che descrive l'interazione con il sistema

Il diagramma in figura 28 rappresenta il flusso principale dell'applicazione MyTalk. Da tale punto è possibile autenticarsi al sistema per poi poter usufruire di tutti i servizi messi a disposizione dal prodotto. Nei successivi diagrammi, figura 30 e 31, vengono descritti rispettivamente le specializzazioni delle attività presenti nella schermata di login: registrazione (se non si ha ancora a disposizione un account utente) e recupero password (nel caso la password sia stata smarrita).

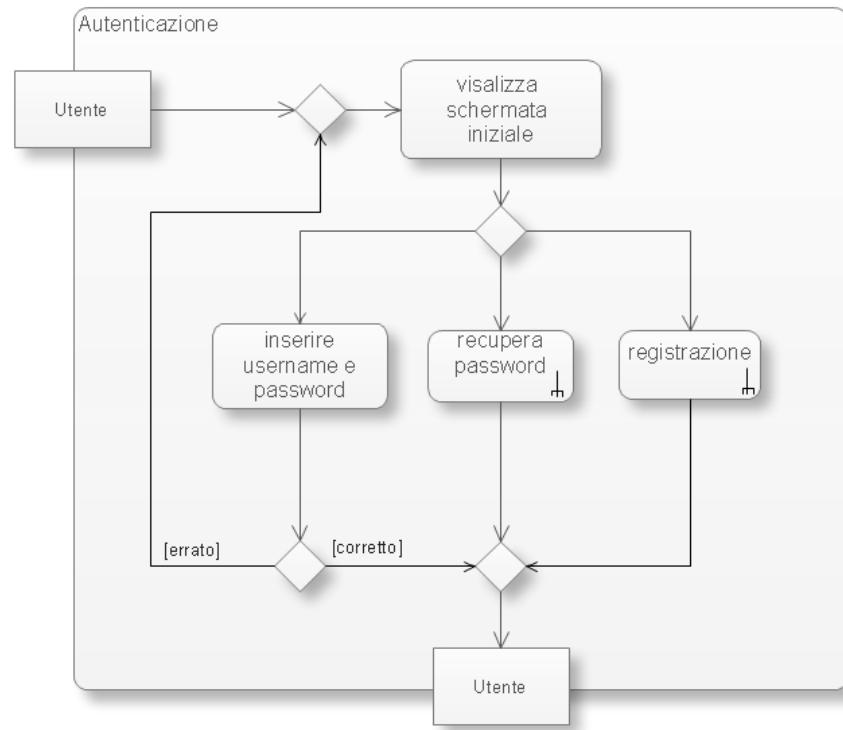


Figura 29: Diagramma di attività relativo all'autenticazione



Figura 30: Diagramma di attività relativo alla registrazione

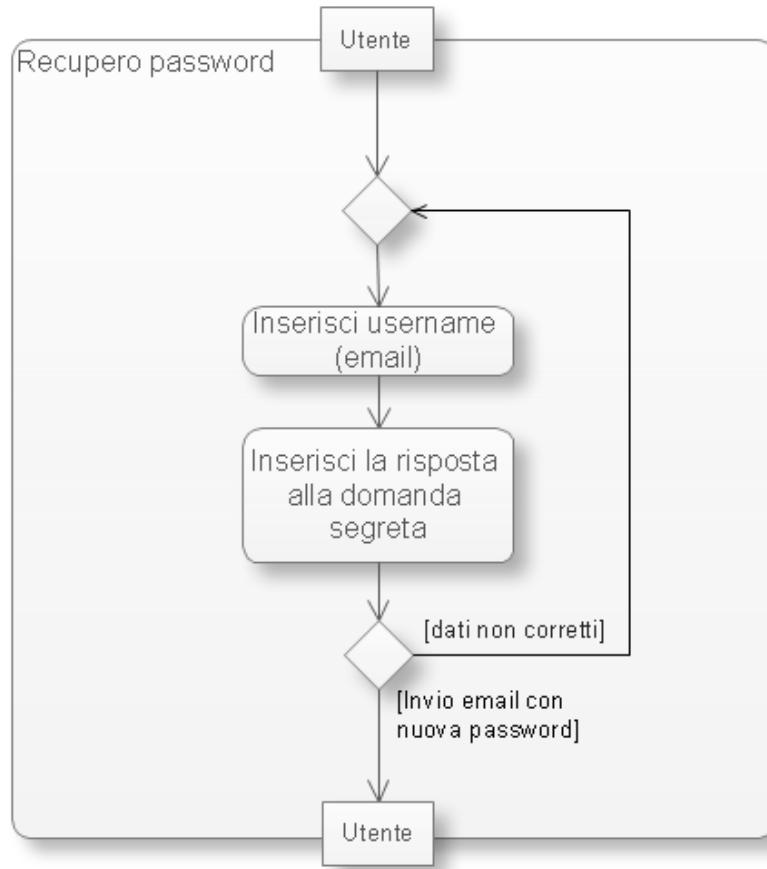


Figura 31: Diagramma di attività relativo al recupero della password

Dal diagramma principale (ricordiamo in figura 28 a pagina 45) una volta autenticati è possibile raggiungere ogni funzionalità del software, come ad esempio la gestione della rubrica (figura 32), la gestione del proprio account (figura 33) o della segreteria (figura 34).

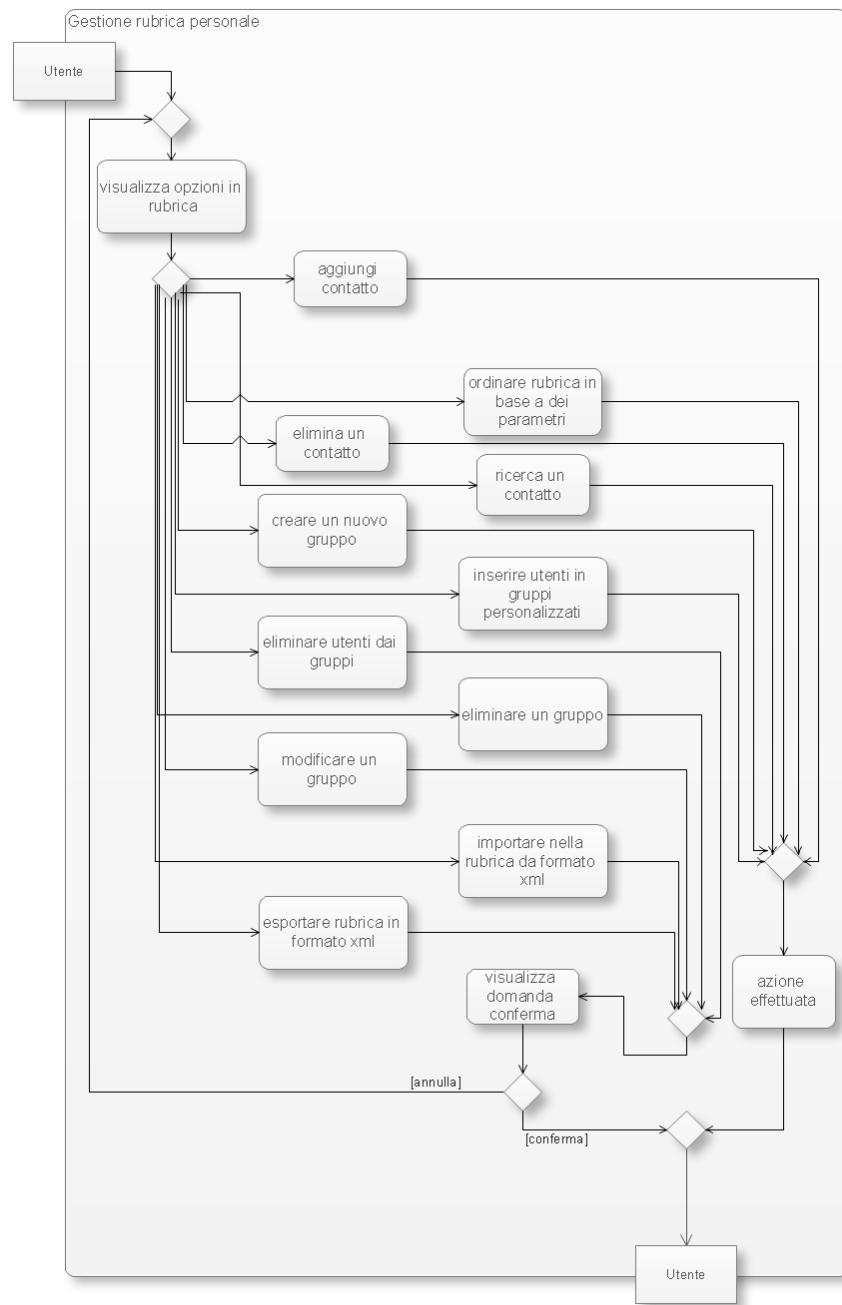


Figura 32: Diagramma di attività relativo alla gestione della rubrica personale

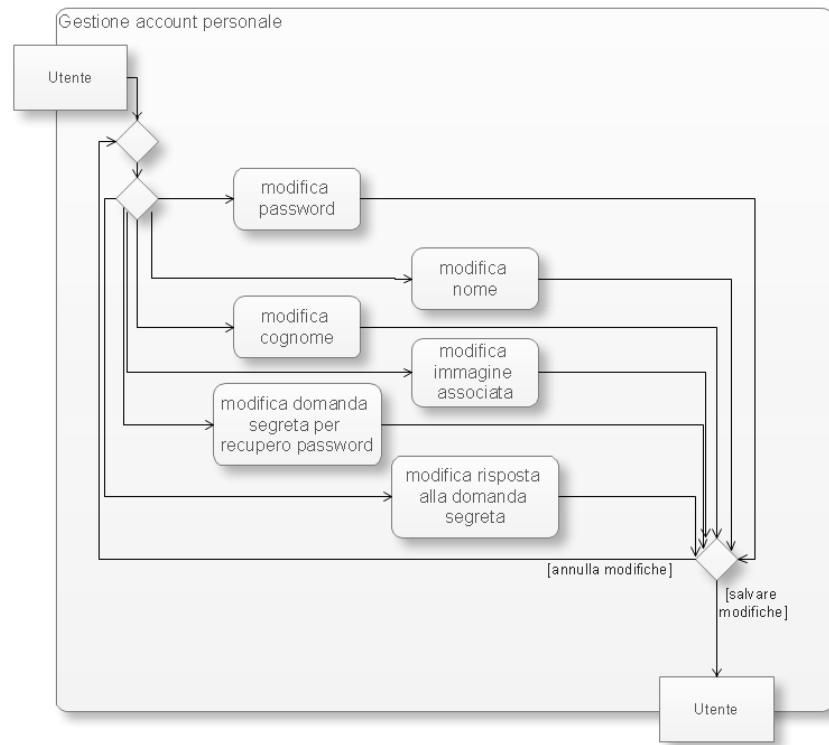


Figura 33: Diagramma di attività relativo alla gestione dei dati dell'account personale

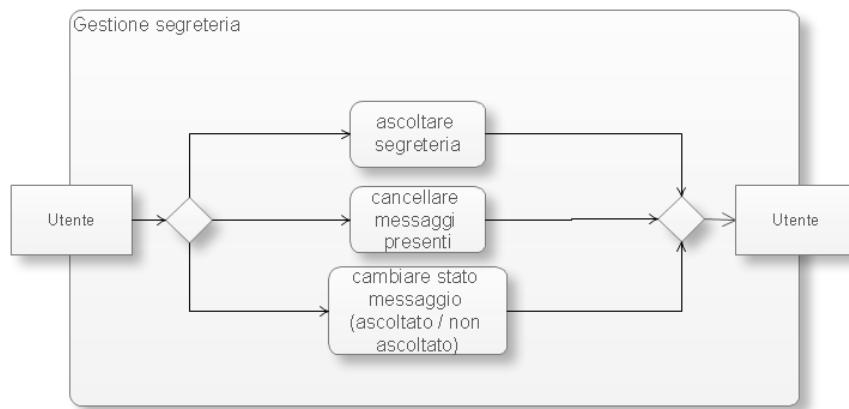


Figura 34: Diagramma di attività relativo alla gestione della segreteria

È infine data la possibilità di connettersi con altri utenti tramite l'attività connessione illustrata nel dettaglio in figura 35.

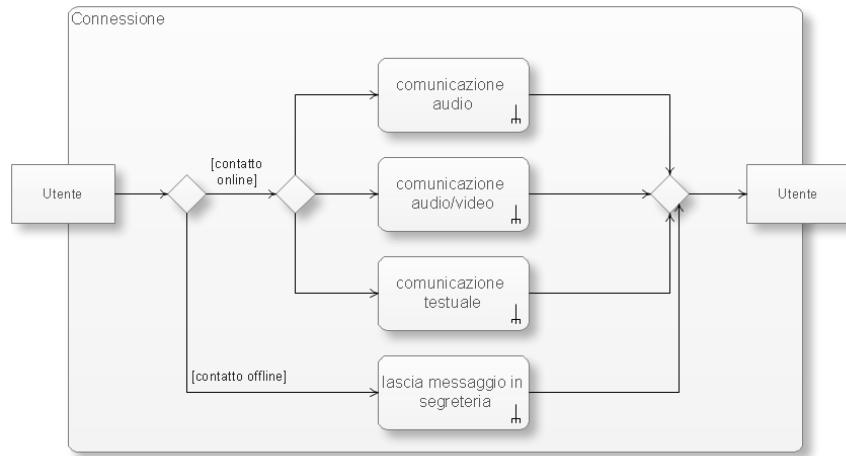


Figura 35: Diagramma di attività relativo alla connessione

La connessione può essere di tipo audio (figura 36), audio/video (figura 37) o testuale (figura 38). In tutte e tre le tipologie di connessione è possibile effettuare una condivisione di risorse, come specificato in figura 39, o registrare una chiamata (tranne nel caso di comunicazione testuale) figura 40.

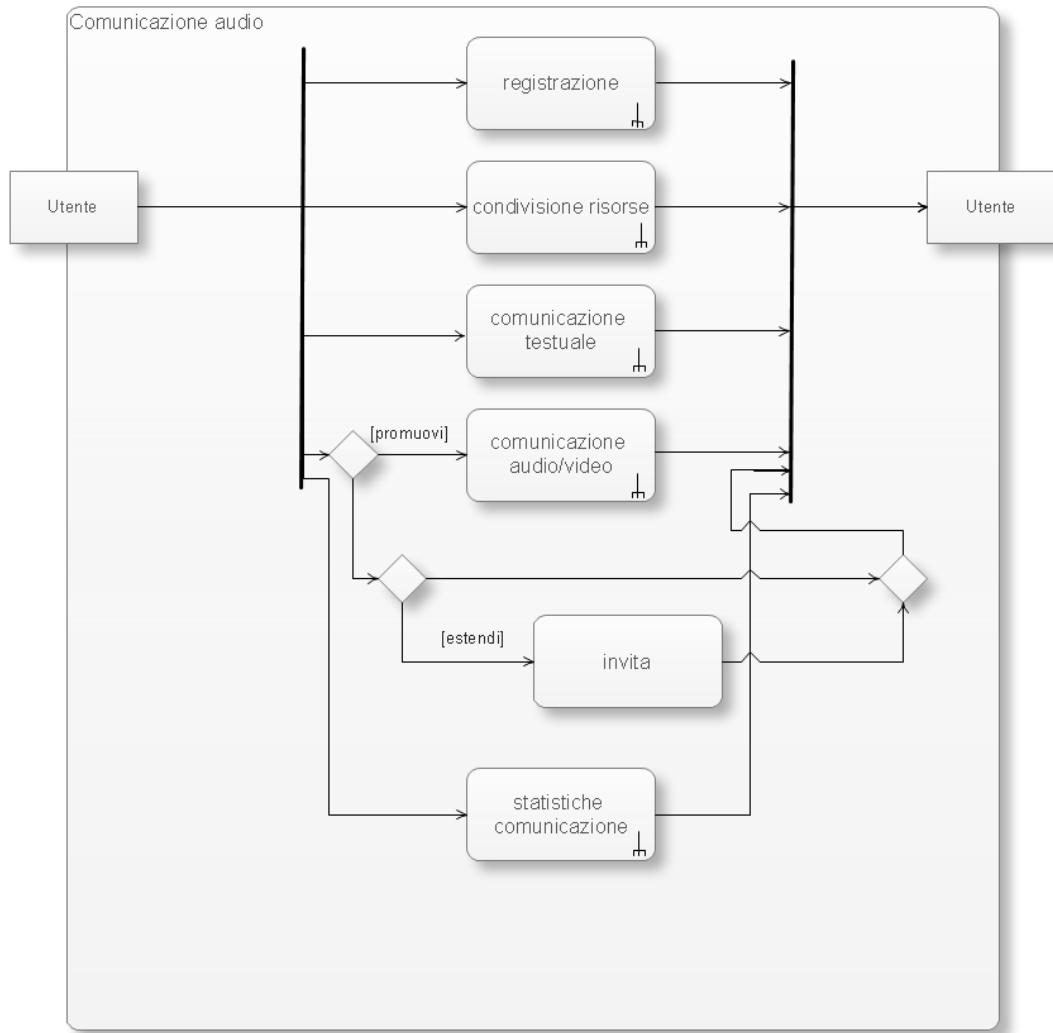


Figura 36: Diagramma di attività relativo alla comunicazione audio

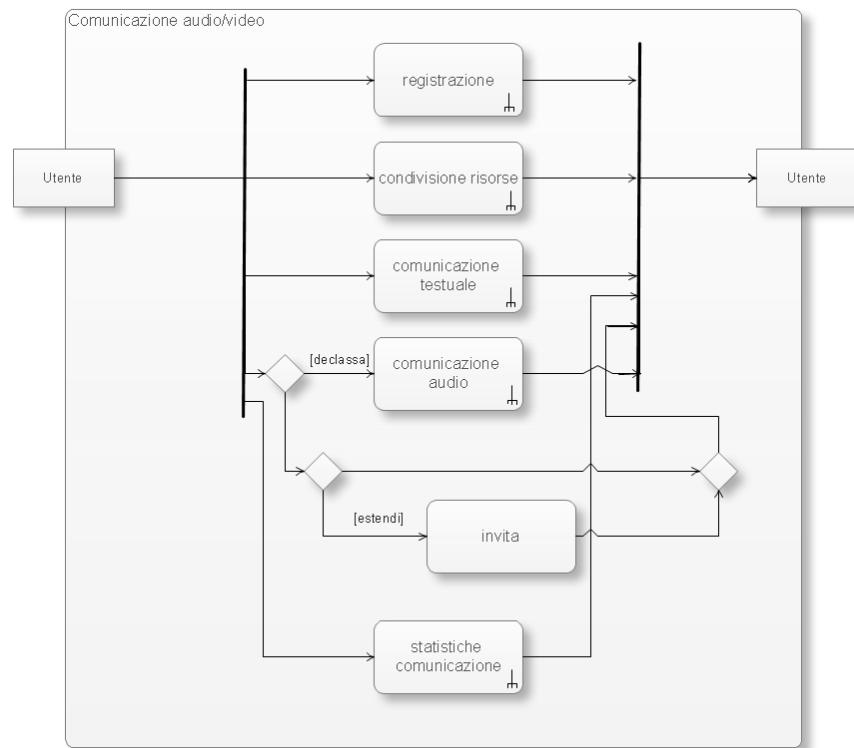


Figura 37: Diagramma di attività relativo alla comunicazione audio/video

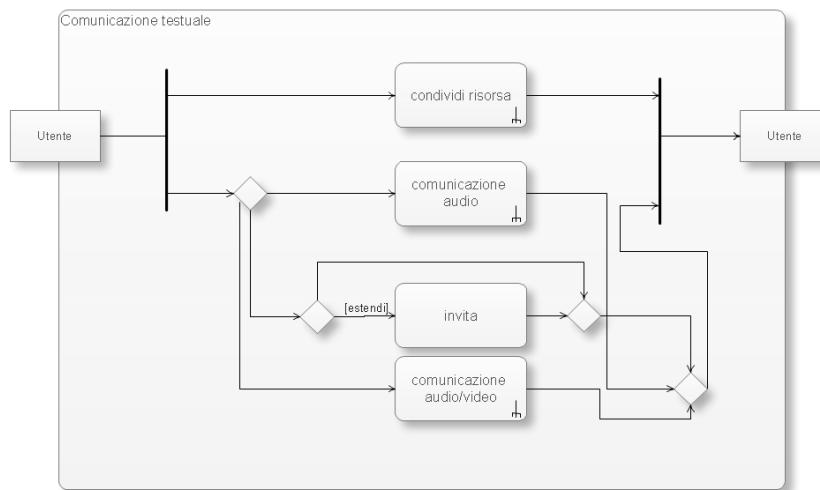


Figura 38: Diagramma di attività relativo alla comunicazione testuale

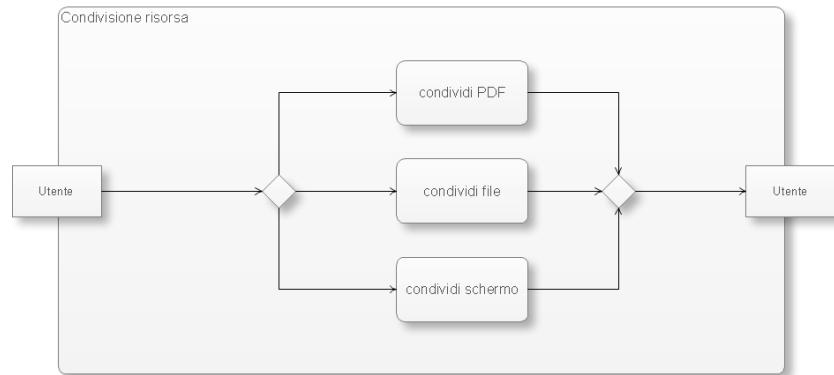


Figura 39: Diagramma di attività relativo alla condivisione di risorse

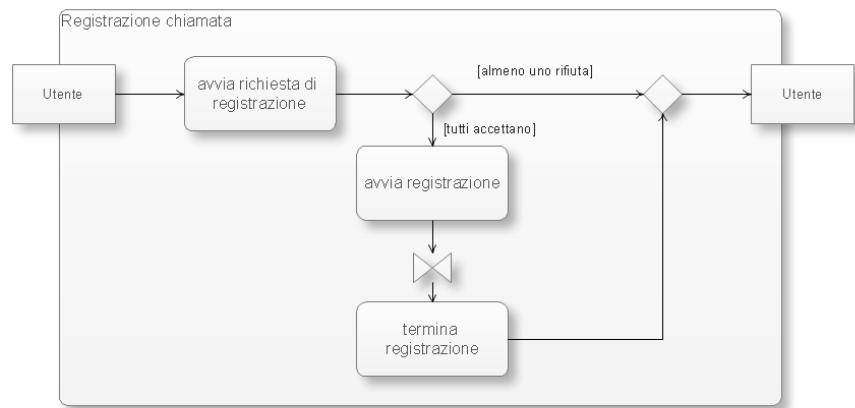


Figura 40: Diagramma di attività relativo alla registrazione della chiamata

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 41.

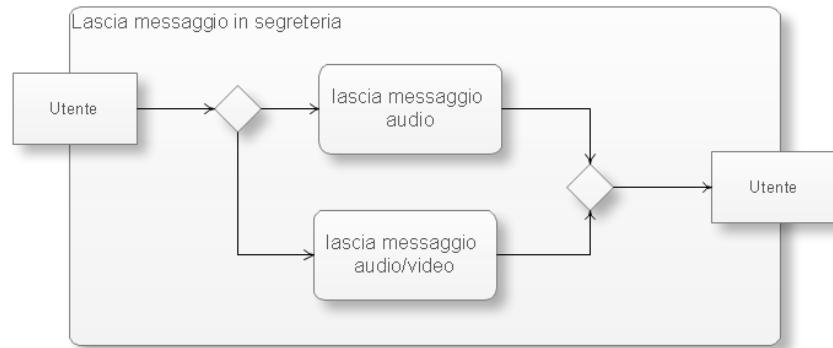


Figura 41: Diagramma di attività relativo alla memorizzazione di un messaggio in segreteria

## 12 Tracciamenti

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

- Dimostrare il soddisfacimento per necessarietà 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.

### 12.1 Tracciamenti Requisiti-Componenti

Requisiti	Componenti
RUFO1.0.0	Façade del server Gestione connessione Gestione database
RUFD1.1.0	Gestione database
RUFD1.1.2	Gestione database
RSQO1.2.0	Gestione connessione Gestione database
RUFO2.0.0	Gestione database Façade del server
RSQO2.1.0	Gestione database
RSDD2.2.0	Gestione database
RUFF3.0.0	Façade del server Gestione database
RUFF3.1.0	Gestione database
RUFF3.2.0	Gestione database
RUFF4.0.0	Gestione rubrica Façade del server Gestione database
RUFF4.1.0	Gestione database Gestione rubrica
RUFF4.2.0	Gestione database Gestione rubrica
RUFF4.3.0	Gestione database Gestione rubrica
RUFF4.4.0	Gestione database Gestione rubrica
RUFF4.4.1	Gestione database Gestione rubrica
RUFF4.4.2	Gestione rubrica Gestione database
RUFF4.4.3	Gestione database Gestione rubrica
RUFF4.4.4	Gestione database Gestione rubrica
RUFF4.5.0	Gestione database Gestione rubrica

RUFF4.6.0	Gestione rubrica
RUFF4.7.0	Gestione database
	Gestione rubrica
RUFO5.0.0	Gestione database
	Façade del server
RUFF5.1.0	Gestione database
RSDO6.0.0	Gestione connessione
RUFO6.1.0	Gestione connessione
	Façade del server
RUFO6.1.1	Gestione connessione
RUFF6.1.2	Gestione connessione
RUFO6.1.3	Gestione connessione
RUFF6.1.4	Gestione connessione
RUFO6.2.0	Façade del server
	Gestione connessione
RUFO6.2.1	Gestione connessione
RUFF6.2.2	Gestione connessione
RUFO6.2.3	Gestione connessione
RUFF6.2.4	Gestione connessione
RUFF6.2.5	Gestione connessione
RUFF6.3.0	Gestione connessione
RUFO7.0.0	Gestione connessione
RUFO8.0.0	Gestione connessione
RUFO8.1.0	Gestione connessione
RUFO8.2.0	Gestione connessione
RUFO9.0.0	Gestione connessione
RUFO9.1.0	Gestione connessione
RUFO9.2.0	Gestione connessione
RUFF9.3.0	Gestione connessione
RSFO11.0.0	Gestione connessione
RSFO11.1.0	Gestione connessione
RSFF11.2.0	Gestione connessione
RSFO12.0.0	Gestione connessione
	Façade del server
RSFF12.1.0	Gestione connessione
RUFD12.2.0	Gestione connessione
RUFO12.3.0	Gestione connessione
RUFF13.0.0	Façade del server
	Gestione connessione
RUFD13.1.0	Gestione connessione
RUFF13.2.0	Gestione connessione
RUFF14.0.0	Gestione connessione
RUFF14.1.0	Gestione connessione
RUFF14.2.0	Gestione connessione
RUFF15.0.0	Gestione segreteria
	Gestione database
	Gestione connessione
	Façade del server
RUFF15.1.0	Gestione segreteria
	Gestione database
RUFF15.2.0	Gestione segreteria
	Gestione connessione

	Gestione database
RUFF15.3.0	Gestione segreteria
	Gestione connessione
	Gestione database
RUFF15.4.0	Gestione database
	Gestione segreteria
	Gestione connessione
RUFF15.5.0	Gestione segreteria
	Gestione connessione
	Gestione database
RUFF16.0.0	Gestione stato
	Gestione connessione
	Façade del server
RUFF17.0.0	Gestione stato
	Gestione connessione
	Façade del server
RUFF18.0.0	Façade del server
	Gestione database
	Gestione connessione
RUFF19.0.0	Gestione connessione
RUFF20.0.0	Gestione connessione
RUFF20.1.0	Gestione connessione
RUFF20.2.0	Gestione connessione
RUFF20.3.0	Gestione connessione
RUFF20.4.0	Gestione connessione

## 12.2 Tracciamenti Componenti-Requisiti

Componenti	Requisiti associati
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
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
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
Gestione stato	RUFF16.0.0
	RUFF17.0.0
Gestione segreteria	RUFF15.5.0
	RUFF15.0.0
	RUFF15.1.0
	RUFF15.2.0
	RUFF15.3.0
	RUFF15.4.0
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

### 12.3 Tracciamenti Componenti-DesignPattern

Componenti	Design pattern utilizzati
Façade del presenter	Façade Singleton
Façade della vista	Singleton Façade

Gestione GUI	Singleton
Gestione database	Data Access Object
Gestione connessione	Factory Method
	Singleton
Gestione rubrica	Proxy
	Composite
Gestione stato	State
	Observer
Gestione segreteria	Proxy
Façade del server	Factory Method
	Singleton
	Façade

## 12.4 Tracciamenti DesignPattern-Componenti

Design pattern	Componenti
Proxy	Gestione rubrica Gestione segreteria
Composite	Gestione rubrica
Data Access Object	Gestione database
Façade	Façade del server Façade del presenter Façade della vista
Factory Method	Gestione connessione Façade del server
Observer	Gestione stato
Singleton	Gestione connessione Façade del server Façade del presenter Façade della vista Gestione GUI
State	Gestione stato

## 12.5 Tracciamenti Componenti-Classi

Componenti	Classi
Gestione comunicazione	StandardClient IServerFacade IConnection IClient
Façade del presenter	IViewFacade IServerFacade IPresenterFacade StandardPresenterFacade
Façade della vista	IViewFacade

Gestione GUI	StandardViewFacade SearchResultPanel ContactPanel MessagePanel LanguagePanel GUIHandler AccountSettingsPanel IPresenterFacade AddressBookPanel CallHistoryPanel MainPanel ToolsPanel
Gestione database	IUserData AudioMessage AudioVideoMessage StandardUserData IAudioMessage IGroup IAudioVideoMessage StandardGroup
Gestione connessione	IState StandardCommunicationHandler IUserData IConnection WebRTCInfo ICommunicationHandler
Gestione rubrica	IUserData UserDataProxy IContact IGroup IAddressBook AddressBook
Gestione stato	IState StateOnline StateOffline StandardUserData StateAvailable StateOccupied
Gestione segreteria	IAudioVideoMessage AudioMessage AudioVideoMessage IMessageBox IAudioMessage AudioMessageProxy StandardMessageBox AudioVideoMessageProxy
Façade del server	IServerFacade StandardServerFacade

## 12.6 Tracciamenti Classi-Componenti

Classi	Componenti
IViewFacade	Façade del presenter
	Façade della vista
AccountSettingsPanel	Gestione GUI
StandardViewFacade	Façade della vista
AudioMessageProxy	Gestione segreteria
IContact	Gestione rubrica
CallHistoryPanel	Gestione GUI
IComunicationHandler	Gestione connessione
StandardComunicationHandler	Gestione connessione
AudioVideoMessageProxy	Gestione segreteria
GUIHandler	Gestione GUI
IAddressBook	Gestione rubrica
IConnection	Gestione connessione
IServerFacade	Gestione comunicazione
IConnection	Gestione comunicazione
IServerFacade	Façade del presenter
AddressBookPanel	Gestione GUI
AddressBook	Gestione rubrica
IServerFacade	Façade del server
StandardServerFacade	Façade del server
WebRTCInfo	Gestione connessione
IState	Gestione stato
	Gestione connessione
MainPanel	Gestione GUI
UserDataProxy	Gestione rubrica
StateOnline	Gestione stato
IPresenterFacade	Gestione GUI
	Façade del presenter
ToolsPanel	Gestione GUI
SearchResultPanel	Gestione GUI
StateOffline	Gestione stato
IMessageBox	Gestione segreteria
StandardPresenterFacade	Façade del presenter
ContactPanel	Gestione GUI
StateAvailable	Gestione stato
StandardMessageBox	Gestione segreteria
MessagePanel	Gestione GUI
IClient	Gestione comunicazione
StateOccupied	Gestione stato
LanguagePanel	Gestione GUI
StandardClient	Gestione comunicazione
IAudioMessage	Gestione database
	Gestione segreteria
IAudioVideoMessage	Gestione segreteria
	Gestione database
AudioMessage	Gestione segreteria
	Gestione database
AudioVideoMessage	Gestione database

	Gestione segreteria
IGroup	Gestione database
	Gestione rubrica
StandardGroup	Gestione database
IUserData	Gestione database
	Gestione rubrica
	Gestione connessione
StandardUserData	Gestione stato
	Gestione database