28 giugno 2013

GoGo Team

Definizione di Prodotto

Informazioni sul documento

Nome Documento	Definizione di Prodotto
Versione	2.0
Stato	*Formale*
Uso	*Esterno*
Data Creazione	04 marzo 2013
Data Ultima Modifica	28 giugno 2013
Redazione	Valentina Pasqualotto
	Francesco Zattarin
Approvazione	Matteo Belletti
Verifica	Sara Lazzaretto
Lista distribuzione	GoGo Team
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Il proponente Zucchetti S.p.A.

Sommario

Il presente documento ha lo scopo di definire i componenti del prodotto **MyTalk**. Per ogni package viene definita la sua struttura delle classi e le interfacce, precedentemente definite nel documento di Specifica Tecnica, oltre ad indicarne le funzionalità, componenti interni, relazioni e dipendenze con altri elementi.

Registro delle modifiche

Versione	Autore	Data	Descrizione
2.0	Valentina Pasqualotto	2013-06-28	Approvazione. Approvazione del documento, cambio di stato in Formale a uso esterno ed avanzamento di versione.
1.3	Matteo Belletti	2013-06-27	Correzione. Correzioni dei contenuti ed ortografiche a seguito delle segnalazioni effettuate dal verificatore Sara Lazzaretto in data 2013-06-26 . Revisione.
1.2	Matteo Belletti	2013-06-26	Aggiunta contenuti. Nuove classi e metodi ed aggiornamento diagrammi $\text{UML}_{\lvert g \rvert}$.
1.1	Valentina Pasqualotto	2013-06-25	Correzioni. Correzione del capitolo *Specifica delle comunicazioni* e dei problemi di layout a seguito delle segnalazioni effettuate in sede di RQ dal *Committente* prof. Tullio Vardanega in data 2013-06-23.
1.0	Valentina Pasqualotto	2013-06-10	Approvazione. Approvazione del documento, cambio di stato in Formale a uso esterno ed avanzamento di versione.
0.7	Matteo Belletti	2013-06-06	Correzione. Correzioni dei contenuti ed ortografiche a seguito delle segnalazioni effettuate dal verificatore Elena Zerbato in data 2013-06-04 . Revisione.
0.6	Elena Zerbato	2013-05-31	Aggiunta contenuti. Diagrammi delle classi nelle relative sezioni di specifica delle componenti e stesura capitolo *Diagrammi di sequenza*.

0.5	Matteo Belletti	2013-05-28	Aggiunta contenuti. Elenco package ad inizio di ogni sezione di specifica delle componenti.
0.4	Matteo Belletti	2013-04-24	Aggiunta contenuti. Capitolo *Specifica della componente View*.
0.3	Davide Ceccon	2013-02-21	Aggiunta contenuti. Capitoli *Specifica della componente Presenter* e *Specifica della componente Model*
0.2	Davide Ceccon	2013-02-21	Aggiunta contenuti. Capitolo *Specifica delle comunicazioni*.
0.1	Davide Ceccon	2013-02-11	Prima stesura. Contenuti documento: scheletro di base dell'itero documento e redatti i capitoli 1 e 2.

Tabella 1: Versionamento del documento

Storico

RP ->RQ

Versione 1.0	Nominativo
Redazione	Matteo Belletti, Davide Ceccon
Verifica	Elena Zerbato
Approvazione	Valentina Pasqualotto

Tabella 2: Storico ruoli RP ->RQ

RQ ->RA

Versione 2.0	Nominativo
Redazione	Valentina Pasqualotto, Francesco Zattarin
Verifica	Sara Lazzaretto
Approvazione	Matteo Belletti

Tabella 3: Storico ruoli RQ ->RA

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di definire in modo approfondito la struttura e le relazioni dei vari componenti del prodotto **MyTalk**, riprendendo e specificando la struttura già definita nel documento di Specifica Tecnica (*SpecificaTecnica_v3.0.pdf*). Questo documento funge da guida e direttiva per i programmatori i quali dovranno implementare i componenti del sistema esattamente come indicato evitando aggiunte o modifiche alle API$_{|g|}$ non previste.

1.2 Scopo del prodotto

Il prodotto **MyTalk** è volto ad offrire la possibilità agli utenti di comunicare tra loro, trasmettendo il segnale audio e video, attraverso il browser$_{|g|}$ mediante l'utilizzo di soli componenti standard, senza che sia necessario installare plugin$_{|g|}$ o programmi aggiuntivi (*es.* Skype). Attualmente, infatti, la comunicazione istantanea tra utenti avviene solo tramite componenti non presenti di default nei browser$_{|g|}$. Il software$_{|g|}$ dovrà risiedere in una singola pagina web$_{|g|}$ e dovrà essere basato sulla tecnologia WebRTC$_{|g|}$ (`http://www.webrtc.org`).

1.3 Norme sul documento corrente

Valgono le seguenti norme relative alla struttura di questo documento:

- Utilizzare esclusivamente i seguenti marcatori d'accesso ad attributi e metodi:
  - \+ per i metodi ed attributi definiti **public**;
  - \# per i metodi ed attributi definiti **protected**;
  - \- per i metodi ed attributi definiti **private**.
- Per definire i dettagli di ogni classe indicata in questo documento dev'essere utilizzata la seguente struttura:
  - Descrizione della funzione svolta dalla classe;
  - Relazioni con altre classi;
  - Elenco attributi della classe, specificandone per ciascuno accessibilità, tipo, nome ed utilizzo;
  - Elenco metodi della classe, specificandone per ciascuno accessibilità, tipo, nome ed utilizzo.

1.4 Glossario

Al fine di migliorare la comprensione al lettore ed evitare ambiguità rispetto ai termini tecnici utilizzati nel documento, viene allegato il file *Glossario_v3.0.pdf*, nel quale vengono descritti i termini contrassegnati dal simbolo $_{|g|}$ alla fine della parola. Per i termini composti da più parole, oltre al simbolo $_{|g|}$, è presente anche la sottolineatura.

1.5 Riferimenti

1.5.1 Normativi

- Capitolato d'appalto: **MyTalk**, *software*$_{|g|}$ *di comunicazione tra utenti senza requisiti di installazione*, rilasciato dal proponente *Zucchetti S.p.A.*, reperibile all'indirizzo http://www.math.unipd.it/~tullio/IS-1/2012/Progetto/C1.pdf.
- Analisi dei requisiti (allegato *AnalisiDeiRequisiti_v4.0.pdf*).
- Norme di progetto (allegato *NormeDiProgetto_v4.0.pdf*).

1.5.2 Informativi

- Specifica Tecnica (allegato *SpecificaTecnica_v3.0.pdf*).

2 Standard di progetto

2.1 Standard di progettazione architetturale

Ciascun diagramma UML$_{|g|}$ inserito nel presente documento è realizzato in conformità con lo standard UML 2.x. Per ulteriori informazioni sugli standard di progettazione architetturale adottati si rimanda ai documenti di Specifica Tecnica (allegato *SpecificaTecnica_v3.0.pdf*) e Norme di Progetto (allegato *NormeDiProgetto_v4.0.pdf*).
Nella comunicazione tra client e server (vedi appendice A) è stato scelto il formato XML$_{|g|}$ per incapsulare i messaggi (anche quelli in formato JSON$_{|g|}$).

2.2 Standard di documentazione del codice

Per gli standard di documentazione del codice utilizzati si fa riferimento al documento Norme di Progetto (allegato *NormeDiProgetto_v4.0.pdf*).

2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi definiti, indipendentemente che siano package, classi, metodi o attributi, devo adottare una denominazione quanto più chiara ed esplicita anche a sacrificio della lunghezza del nome. Sono ammesse abbreviazioni purché siano non ambigue e di immediata comprensione.
Per ulteriori informazioni si rimanda al documento Norme di Progetto (allegato *NormeDiProgetto_v4.0.pdf*).

2.4 Standard di programmazione

Gli standard di programmazione sono definiti nel documento Norme di Progetto (allegato *NormeDiProgetto_v4.0.pdf*) al quale si rimanda per ulteriori informazioni.

2.5 Strumenti di lavoro

Gli strumenti di lavoro adottati sono quelli individuati e descritti nel documento Norme di Progetto (allegato *NormeDiProgetto_v4.0.pdf*).

3 Specifica della componente View

La componente View si occupa di interagire con l'utente mettendogli a disposizione un'interfaccia ($GUI_{|g|}$) tramite la quale egli può immettere dati, visualizzare informazioni relative al proprio profilo e gestire le chiamate. La View invia al Presenter le richieste di elaborazione dei dati e riceve da quest'ultimo le richieste di aggiornamento dell'interfaccia utente.
È formata dalle classi:

`mytalk.client.iView.iUser.ICallManage` (sez. 3.1.1)

`mytalk.client.iView.iUser.ICommunicationView` (sez. 3.1.2)

`mytalk.client.iView.iUser.ILogUser` (sez. 3.1.3)

`mytalk.client.iView.iUser.IMakeCall` (sez. 3.1.4)

`mytalk.client.iView.iUser.IMediaManage` (sez. 3.1.5)

`mytalk.client.iView.iUser.IModifyDataUser` (sez. 3.1.6)

`mytalk.client.iView.iUser.IPageUserView` (sez. 3.1.7)

`mytalk.client.iView.iUser.IRegister` (sez. 3.1.8)

`mytalk.client.iView.iUser.IShowDataUser` (sez. 3.1.9)

`mytalk.client.iView.iUser.IUserDataView` (sez. 3.1.10)

`mytalk.client.view.user.CallManage` (sez. 3.2.1)

`mytalk.client.view.user.CommunicationView` (sez. 3.2.2)

`mytalk.client.view.user.LogUser` (sez. 3.2.3)

`mytalk.client.view.user.MakeCall` (sez. 3.2.4)

`mytalk.client.view.user.MediaManage` (sez. 3.2.5)

`mytalk.client.view.user.ModifyDataUser` (sez. 3.2.6)

`mytalk.client.view.user.PageUserView` (sez. 3.2.7)

`mytalk.client.view.user.Register` (sez. 3.2.8)

`mytalk.client.view.user.ShowDataUser` (sez. 3.2.9)

`mytalk.client.view.user.UserDataView` (sez. 3.2.10)

`mytalk.client.iView.iAdministrator.ILogAdmin` (sez. 3.3.1)

`mytalk.client.iView.iAdministrator.IPageAdminView` (sez. 3.3.2)

`mytalk.client.iView.iAdministrator.IStatisticView` (sez. 3.3.3)

`mytalk.client.view.administrator.LogAdmin` (sez. 3.4.1)

`mytalk.client.view.administrator.PageAdminView` (sez. 3.4.2)

`mytalk.client.view.administrator.StatisticView` (sez. 3.4.3)

3.1 Package mytalk.client.iView.iUser

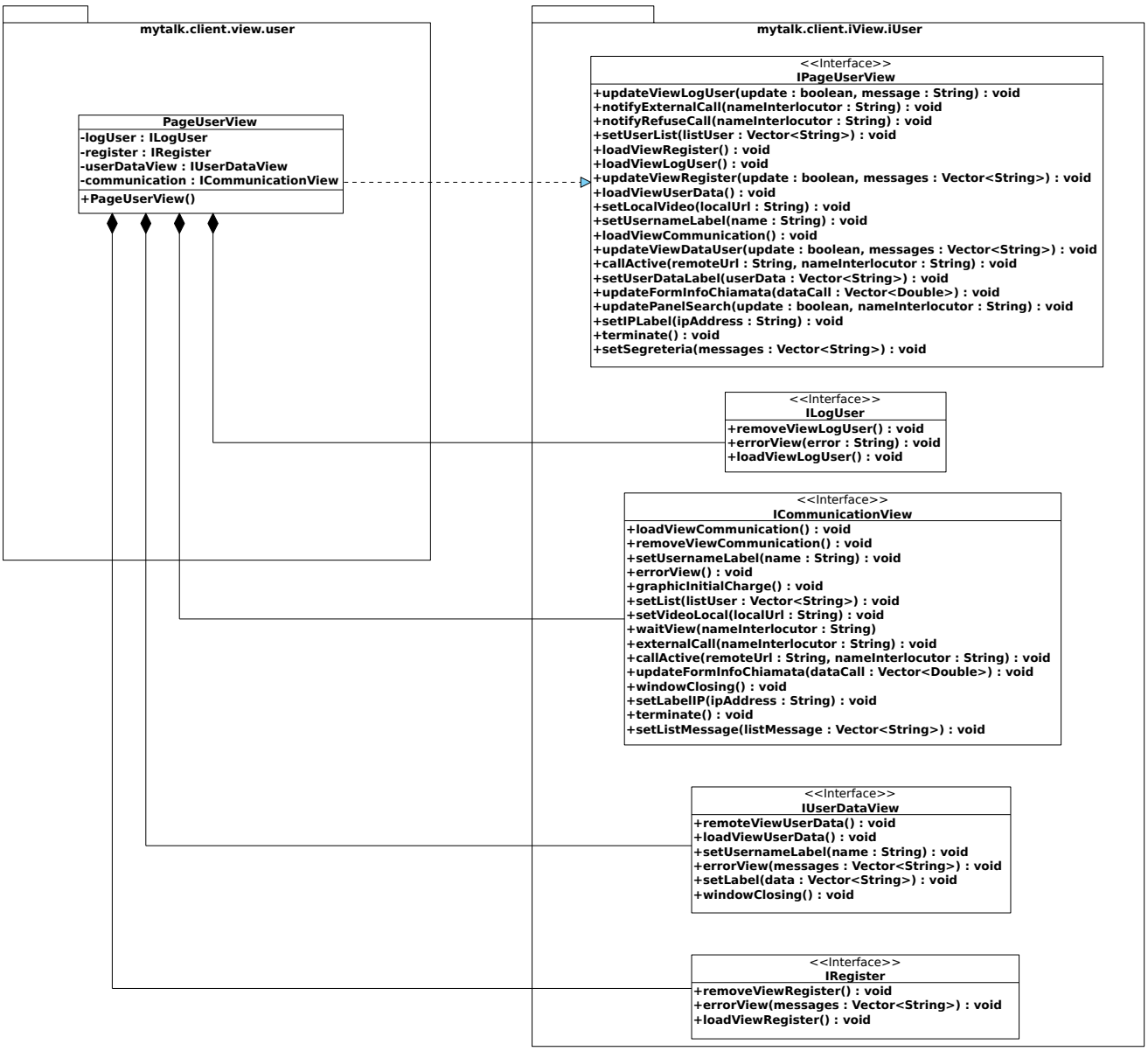

Figura 1: Diagramma delle classi dei package `mytalk.client.iView.iUser` e `mytalk.client.view.user`; dettaglio delle classi `IPageUserView`, `ILogUser`, `ICommunicationView`, `IUserDataView`, `IRegister` e `PageUserView`.

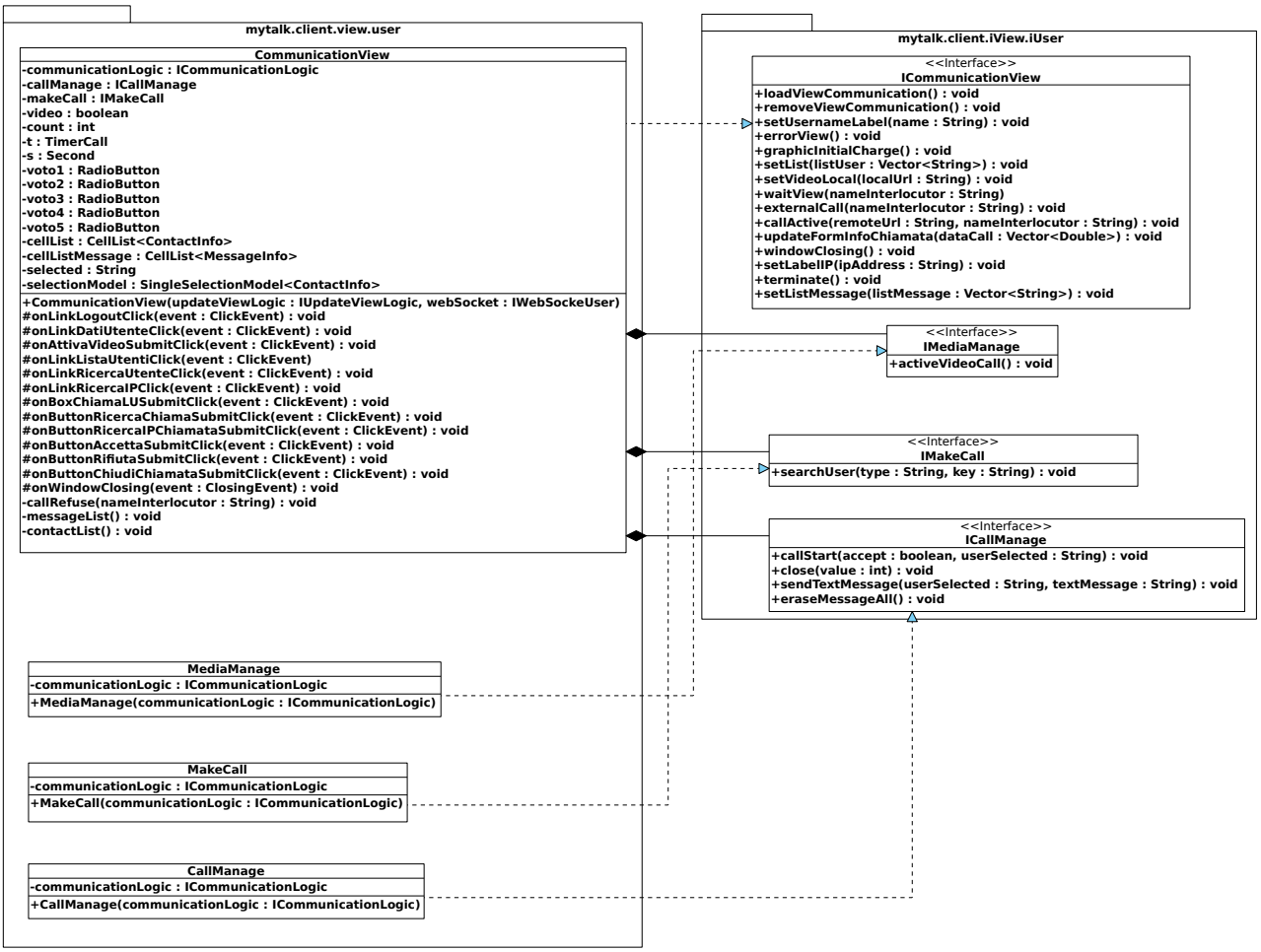

Figura 2: Diagramma delle classi dei package `mytalk.client.iView.iUser` e `mytalk.client.view.user`; dettaglio delle classi `ICommunicationView`, `IMediaManage`, `IMakeCall`, `ICallManage`, `CommunicationView`, `MediaManage`, `MakeCall` e `CallManage`.

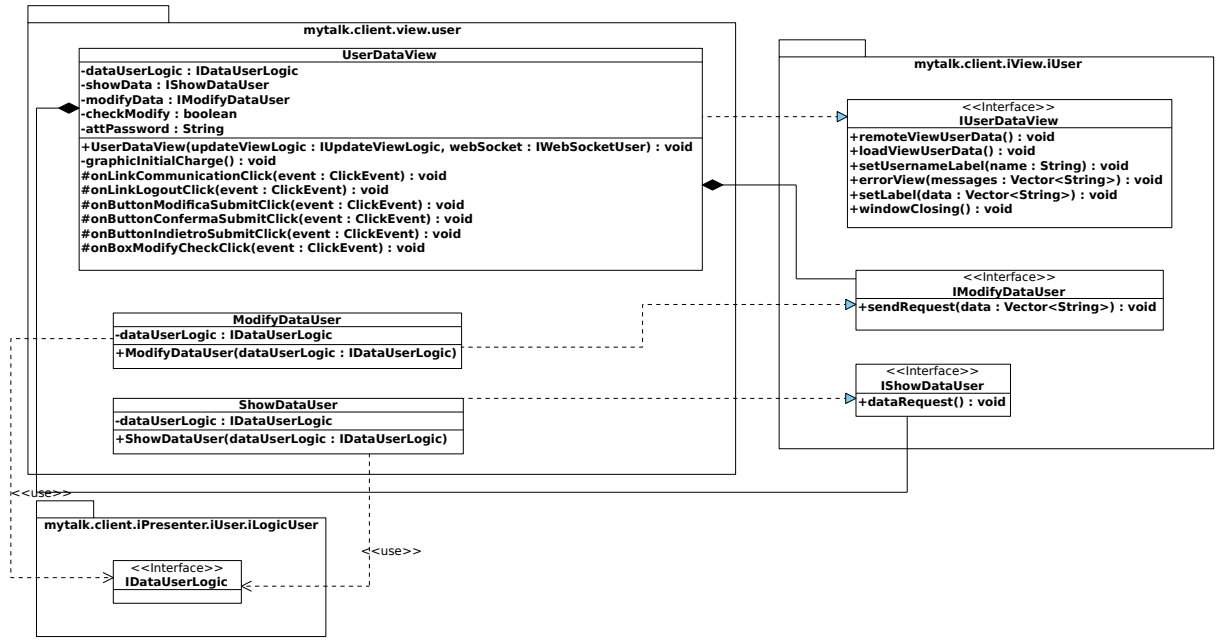

Figura 3: Diagramma delle classi dei package `mytalk.client.iView.iUser` e `mytalk.client.view.user`; dettaglio delle classi `IUserDataView`, `IModifyDataUser`, `IShowDataUser`, `UserDataView`, `ModifyDataUser` e `ShowDataUser`.

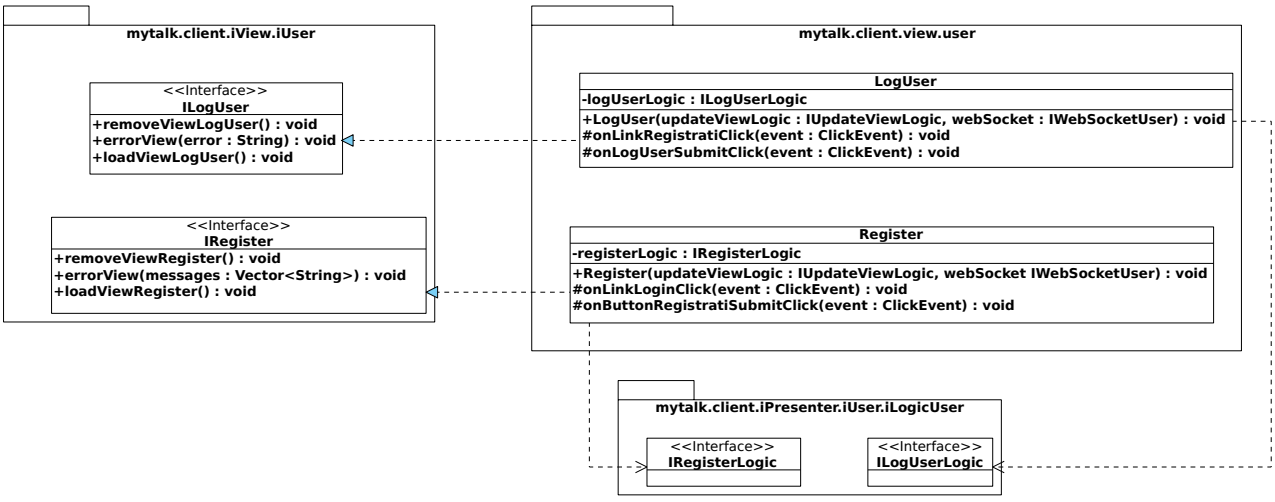

Figura 4: Diagramma delle classi dei package `mytalk.client.iView.iUser` e `mytalk.client.view.user`; dettaglio delle classi `ILogUser`, `IRegister`, `LogUser` e `Register`.

3.1.1 ICallManage

Funzione:
Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ per la gestione di una comunicazione attiva e interagiscono con essa. L'interfaccia offre inoltre i metodi per l'uso della segreteria.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.CallManage`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.CommunicationView`.

Metodi:
`+ void callStart(boolean accept, String userSelected);`
Invia al Presenter, a seconda del valore `accept`, la richiesta di accettazione o di rifiuto di una comunicazione.

`+ void close(int value);`
Invia una richiesta al Presenter per chiudere la comunicazione attiva, inviando il valore della valutazione dell'utente.

`+ void sendTextMessage(String userSelected, String textMessage);`
Invia una richiesta al Presenter di invio del messaggio di segreteria verso l'utente selezionato.

`+ void eraseMessageAll();`
Invia una richiesta al presenter per eliminare tutti i messaggi registrati nella segreteria.

3.1.2 ICommunicationView

Funzione:
Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ della comunicazione e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.CommunicationView`.

L'interfaccia è utilizzata da

`mytalk.client.view.user.PageUserView`.

Metodi:
`+ void loadViewCommunication();`
Visualizza la $GUI_{|g|}$ che permette all'utente di gestire le comunicazioni.

`+ void removeViewCommunication();`
Nasconde la $GUI_{|g|}$ di comunicazione.

`+ void setUsenameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'utente autenticato.

`+ void errorView();`
Imposta e rende visibili i messaggi di errore da segnalare all'utente.

`+ void graphicInitialCharge();`
Imposta i pannelli da visualizzare e le impostazioni di default.

`+ void setList(Vector<String> listUser);`
Inserisce gli utenti registrati contenuti nel vettore `listUser` all'interno del pannello `ListUser`.

`+ void setVideoLocal(String localUrl);`
Imposta il video locale con l'indirizzo contenuto nella stringa `localUrl`.

`+ void waitView(String nameInterlocutor);`
Visualizza il pannello, per il mittente, in attesa di risposta del destinatario.

`+ void externalCall(String nameInterlocutor);`
Visualizza il pannello di chiamata in entrata.

`+ void callActive(String remoteUrl, String nameInterlocutor);`
Visualizza il pannello di informazioni di chiamata ed imposta il video remoto con la stringa `remoteUrl` e con il nome dell'utente (parametro `nameInterlocutor`) con cui si è instaurata la comunicazione.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Visualizza le statistiche delle chiamate contenute nel vettore `dataCall`.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ di comunicazione effettuando il logout e in caso di comunicazione attiva termina la chiamata.

`+ void setLabelIP(String ipAddress);`
Imposta l'etichetta `LabelIP` con l'indirizzo $IP_{|g|}$ dell'utente autenticato.

`+ void terminate();`
Metodo che cattura il valore della valutazione dell'utente riguardo la comunicazione in corso. Invia una richiesta di chiusura della comunicazione al Presenter ed, infine, richiama il metodo per tornare alla grafica iniziale.

`+ void setListMessage(Vector<String> listMessage);`
Inserisce i messaggi di segreteria contenuti nel vettore `listMessage` all'interno del pannello `ListMessage`.

3.1.3 ILogUser

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per l'autenticazione degli utenti e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.LogUser`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.PageUserView`.

Metodi:
`+ void loadViewLogUser();`
Visualizza la $GUI_{|g|}$ di autenticazione e seleziona le impostazioni di $default_{|g|}$, i campi vuoti e l'errore (non visibile).

`+ void removeViewLogUser();`
Nasconde la $GUI_{|g|}$ di autenticazione.

`+ void errorView(String error);`
Imposta e rende visibile il messaggio di errore passato dal parametro `error`.

3.1.4 IMakeCall

Funzione: Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ per la gestione della negoziazione di una comunicazione e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.MakeCall`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.CommunicationView`.

Metodi:
`+ void searchUser(String type, String key);`
Invia una richiesta al Presenter per cercare e controllare la disponibilità dell'utente desiderato.

3.1.5 IMediaManage

Funzione: Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ per la gestione dei media per la comunicazione e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.MediaManage`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.CommunicationView`.

Metodi:
`+ void activeVideoLocal();`
Invia una richiesta al Presenter per recuperare l'indirizzo del video locale.

3.1.6 IModifyDataUser

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per la modifica dei dati di un utente e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.ModifyDataUser`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.UserDataView`.

Metodi:
`+ void sendRequest(Vector<String> data);`
Invia una richiesta per sostituire i vecchi dati utente con quelli contenuti nel vettore `data`.

3.1.7 IPageUserView

Funzione:
Interfaccia che offre operazioni alle classi che compongono la GUI$_{|g|}$ e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.PageUserView`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`;

`mytalk.client.presenter.user.logicUser.UpdateUserView`.

Metodi:
`+ void updateViewLogUser(boolean update, String message);`
Aggiorna la GUI$_{|g|}$ a seconda del valore `update`:

- se `false`: aggiorna la GUI$_{|g|}$ di autenticazione visualizzando l'errore opportuno contenuto in `message`;
- se `true`: rende invisibile la GUI$_{|g|}$ di autenticazione e visualizza la GUI$_{|g|}$ di comunicazione.

`+ void notifyExternalCall(String nameInterlocutor);`
Notifica alla GUI$_{|g|}$ di comunicazione l'arrivo di una richiesta di chiamata esterna.

`+ void notifyRefuseCall(String nameInterlocutor);`
Notifica alla GUI$_{|g|}$ di comunicazione il rifiuto di una richiesta di chiamata.

`+ void setUserList(Vector<String> listUser);`
Imposta la lista dei possibili utenti (parametro `listUser`) con i quali instaurare una chiamata nella GUI$_{|g|}$ di comunicazione.

`+ void loadViewRegister();`
Richiede la visualizzazione della GUI$_{|g|}$ di registrazione.

`+ void loadViewLogUser();`
Richiede la visualizzazione della $GUI_{|g|}$ di autenticazione.

`+ void updateViewRegister(boolean update, Vector<String> messages);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: aggiorna la $GUI_{|g|}$ di registrazione visualizzando gli errori opportuni contenuti in `messages`.
- se `true`: rende invisibile la $GUI_{|g|}$ di registrazione e visualizza la $GUI_{|g|}$ di autenticazione.

`+ void loadViewUserData();`
Richiede la visualizzazione della $GUI_{|g|}$ di gestione dei dati dell'utente.

`+ void setLocalVideo(String localUrl);`
Invia l'indirizzo del video locale (parametro `localUrl`) alla $GUI_{|g|}$ di comunicazione.

`+ void setUsernameLabel(String name);`
Invia il nome utente (parametro `name`) alla $GUI_{|g|}$ di comunicazione e alla $GUI_{|g|}$ di gestione dei dati dell'utente.

`+ void loadViewCommunication();`
Richiede la visualizzazione della $GUI_{|g|}$ di comunicazione.

`+ void updateViewDataUser(boolean update, Vector<String> messages);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: aggiorna la $GUI_{|g|}$ di modifica dei dati dell'utente visualizzando gli errori opportuni contenuti in `messages`;
- se `true`: visualizza il pannello di visualizzazione dei dati dell'utente.

`+ void setUserDataLabel(Vector<String> userData);`
Invia l'elenco dei dati dell'utente alla $GUI_{|g|}$ di visualizzazione dei dati.

`+ void callActive(String remoteUrl, String nameInterlocutor);`
Notifica alla $GUI_{|g|}$ di comunicazione l'inizio di una conversazione, inviandole l'indirizzo del video remoto (parametro `remoteUrl`) e lo username (parametro `nameInterlocutor`) dell'utente remoto.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Invia i dati delle statistiche alla $GUI_{|g|}$ di comunicazione.

`+ void updatePanelSearch(boolean update, String nameInterlocutor);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: visualizza il pannello di attesa di risposta da parte dell'utente in fase di negoziazione;
- se `true`: aggiorna la $GUI_{|g|}$ di comunicazione visualizzando l'errore opportuno.

`+ void setIPLabel(String ipAddress);`
Invia l'indirizzo $IP_{|g|}$ (parametro `ipAddress`) alla $GUI_{|g|}$ di comunicazione.

`+ void terminate();`
Invia alla $GUI_{|g|}$ di comunicazione un segnale di richiesta di chiusura di conversazione.

`+ void setSegreteria(Vector<String> messagges);`
Imposta la lista dei messaggi in segreteria (parametro `messagges`) nella $GUI_{|g|}$ di comunicazione.

3.1.8 IRegister

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per la registrazione di nuovi utenti e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.Register`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.PageUserView`.

Metodi:
`+ void loadViewRegister();`
Visualizza la $GUI_{|g|}$ di registrazione.

`+ void removeViewRegister();`
Nasconde la $GUI_{|g|}$ di registrazione.

`+ void errorView(Vector<String> messages);`
Imposta e rende visibili i messaggi di errore, definiti nel vettore `messages`, da segnalare all'utente.

3.1.9 IShowDataUser

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per la visualizzazione dei dati di un utente e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.ShowDataUser`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.UserDataView`.

Metodi:
`+ void dataRequest();`
Invia una richiesta al Presenter per recuperare i dati relativi all'utente.

3.1.10 IUserDataView

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per la gestione dei dati di un utente e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.user.UserDataView`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.PageUserView`.

Metodi:
`+ void loadViewUserData();`
Visualizza la $GUI_{|g|}$ che permette all'utente di visualizzare i propri dati.

`+ void removeViewUserData();`
Nasconde la $GUI_{|g|}$ di visualizzazione e modifica dati.

`+ void errorView(Vector<String> messages);`
Imposta e rende visibili i messaggi di errore, definiti nel vettore `messages`, da segnalare all'utente.

`+ void setUsenameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'utente autenticato.

`+ void setLabel(Vector<String> data);`
Imposta i campi dei dati dell'utente estrapolati dal vettore `data`.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ di gestione dei dati utente effettuando il logout.

3.2 Package mytalk.client.view.user

3.2.1 CallManage

Funzione: La classe ha il compito di inoltrare al Presenter le richieste per la gestione di una comunicazione attiva.

Relazioni con altre componenti:
Implementa l'interfaccia:

- `mytalk.client.iView.iUser.ICallManageUser`.

Usa le classi:

- `mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Tramite le interfacce:

- `mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic`.

Attributi:
`- ICommunicationLogic communicationLogic`: riferimento alla classe `CommunicationLogic`.

Metodi:
`+ CallManage(IComunicationLogic communicationLogic);`
Costruttore: inizializza `communicationLogic` con il valore ricevuto come parametro.

`+ void callStart(boolean accept, String userSelected);`
Invia al Presenter, a seconda del valore `accept`, la richiesta di accettazione o di rifiuto di una comunicazione. Se `accept` è `true` richiama, attraverso il riferimento `communicationLogic`, il metodo `accept(userSelected)`. In caso contrario richiama `refuse(userSelected)`.

`+ void close(int value);`
Invia una richiesta al Presenter per chiudere la comunicazione attiva, inviando il valore della valutazione dell'utente.

`+ void sendTextMessage(String userSelected, String textMessage);`
Invia una richiesta al Presenter per inviare il messaggio inserito nel parametro `textMessage` all'utente indicato nel parametro `userSelected`.

`+ void eraseMessageAll();`
Invia una richiesta al Presenter per eliminare tutti i messaggi presenti in segreteria.

3.2.2 CommunicationView

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di comunicazione;
- gestire gli eventi che l'utente o il sistema possono innescare: inoltra le richieste di comunicazione all'interfaccia `IMakeCall`, gestisce gli eventi di comunicazione attraverso `ICallManage` e inoltra gli eventi che gestiscono il video e l'audio a `IMediaManage`;
- gestire i link presenti nella pagina.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.ICommunicationView`.

Usa le classi:

`mytalk.client.view.user.MediaManage`;

`mytalk.client.view.user.CallManage`;

`mytalk.client.view.user.MakeCall`;

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Tramite le interfacce:

`mytalk.client.iView.iUser.IMediaManage`;

`mytalk.client.iView.iUser.ICallManage`;

`mytalk.client.iView.iUser.IMakeCall`;

`mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic`.

Attributi:

- `ICommunicationLogic communicationLogic`: riferimento alla classe `CommunicationLogic`.

- `ICallManage callManage`: riferimento alla classe `CallManager`.

- `IMakeCall makeCall`: riferimento alla classe `MakeCall`.

- `IMediaManage mediaManage`: riferimento alla classe `MediaManager`.

- `boolean video`: valore booleano che indica lo stato di attivazione del video.

- `int count`: valore per contare il numero di richieste di conversazione.

- `TimerCall t`: timer che effettua un conto alla rovescia per l'attesa della risposta al momento dell'invio di una richiesta di comunicazione.

- `Second s`: timer che simula la funzione di un cronometro durante una comunicazione.

- `RadioButton voto1`: radio button per assegnare un giudizio di 1.

- `RadioButton voto2`: radio button per assegnare un giudizio di 2.

- `RadioButton voto3`: radio button per assegnare un giudizio di 3.

- `RadioButton voto4`: radio button per assegnare un giudizio di 4.

- `RadioButton voto5`: radio button per assegnare un giudizio di 5.

- `CellList<ContactInfo> cellList`: oggetto lista di celle contenente le informazioni dei contatti.

- `CellList<MessageInfo> cellListMessage`: oggetto lista di celle contenente le informazioni dei messaggi.

- `String selected`: stringa contenente lo username dell'utente attualmente selezionato, se nessun utente è selezionato `selected` risulta vuota.

- `SingleSelectionModel<ContactInfo> selectionModel`: oggetto che indica l'elemento dei `ContactInfo` attualmente selezionato.

Oggetti:
`@UiField HTMLPanel MyDiv`: pannello contenente `DivVideoCommunication` e `DivPanelControl`.

`@UiField HTMLPanel DivVideoCommunication`: pannello contenente `DivVideo`, `DivVideoLocal` e `FormActivateLocalVideo`.

`@UiField HTML DivVideo`: codice $\text{HTML}_{|g|}$ per la grafica del video remoto, attivo e inattivo.

`@UiField HTML DivVideoLocal`: codice $\text{HTML}_{|g|}$ per la grafica del video locale, attivo e inattivo.

`@UiField HTMLPanel FormActivateLocalVideo`: pannello contenente `AttivaVideoSubmit`.

`@UiField PushButton AttivaVideoSubmit`: bottone per l'attivazione del video locale.

`@UiField HTMLPanel DivPanelControl`: pannello contenente `LinkDatiUtente`, `LinkLogout`, `LabelUser`, `TabBar`, `FormListaUtenti`, `FormRicercaUtente`, `FormRicercaIP`, `FormMessaggi`, `FormRicezioneChiamata`, `FormInfoChiamata` e `CaptionChiamataInCorso`.

`@UiField InlineHyperlink LinkDatiUtente`: link per passare alla $\text{GUI}_{|g|}$ di gestione dati utente.

`@UiField InlineHyperlink LinkLogout`: link per passare alla $\text{GUI}_{|g|}$ dell'autenticazione effettuando il logout.

`@UiField InlineLabel LabelUser`: label che visualizza lo username dell'utente autenticato.

`@UiField Grid TabBar`: griglia contenente `LinkListaUtenti`, `LinkRicercaUtente`, `LinkRicercaIP` e `LinkMessaggi`.

`@UiField InlineHyperlink LinkListaUtenti`: link per visualizzare `FormListaUtenti`.

`@UiField HTMLPanel FormListaUtenti`: pannello contenente `ListUser`, `BoxChiamaLUSubmit` e `ButtonSegreteria`.

`@UiField HTMLPanel ListUser`: pannello contenente l'oggetto `cellList`.

`@UiField PushButton BoxChiamaLUSubmit`: bottone per l'invio della richiesta di comunicazione con relativo controllo di stato dell'utente.

`@UiField PushButton ButtonSegreteria`: bottone per l'invio della richiesta di invio di un messaggio di segreteria verso l'utente selezionato.

`@UiField InlineHyperlink LinkRicercaUtente`: link per visualizzare `FormRicercaUtente`.

`@UiField FormPanel FormRicercaUtente`: form contenente `BoxEMail`, `LabelErroreRicerca` e `ButtonRicercaChiamaSubmit`.

`@UiField TextBox BoxEMail`: campo per l'inserimento dell'e-mail dell'utente con il quale si desidera instaurare una comunicazione.

`@UiField InlineLabel LabelErroreRicerca`: label che contiene l'errore relativo alla ricerca dell'utente.

`@UiField PushButton ButtonRicercaChiamaSubmit`: bottone per l'invio della richiesta di comunicazione con relativo controllo di esistenza e di stato.

`@UiField InlineHyperlink LinkRicercaIP`: link per visualizzare la `FormRicercaIP`.

`@UiField FormPanel FormRicercaIP`: form contenente `LabelIP`, `BoxRicercaIP`, `LabelErroreRicercaIP` e `ButtonRicercaIPChiamaSubmit`.

`@UiField InlineLabel LabelIP`: label che contiene l'attuale indirizzo $IP_{|g|}$ dell'utente.

`@UiField TextBox BoxRicercaIP`: campo per l'inserimento dell'indirizzo $IP_{|g|}$ dell'utente con il quale si desidera instaurare una comunicazione.

`@UiField InlineLabel LabelErroreRicercaIP`: label che contiene l'errore relativo alla ricerca dell'utente tramite indirizzo $IP_{|g|}$.

`@UiField PushButton ButtonRicercaIPChiamaSubmit`: bottone per l'invio della richiesta di comunicazione con relativo controllo di esistenza e di stato tramite indirizzo $IP_{|g|}$.

`@UiField HTMLPanel FormMessaggi`: pannello contenente `LabelMessaggi`, `ListMessage` e `ButtonEliminaTutti`.

`@UiField HTMLPanel ListMessage`: pannello contenente l'oggetto `cellListMessage`.

`@UiField PushButton ButtonEliminaTutti`: bottone per l'invio della richiesta dell'eliminazione di tutti i messaggi contenuti nella segreteria.

`@UiField FormPanel FormRicezioneChiamata`: form contenente `LabelNomeUtente`, `ButtonAccettaSubmit` e `ButtonRifiutaSubmit`.

`@UiField InlineLabel LabelNomeUtente`: label che contiene il nome del mittente della richiesta di chiamata.

`@UiField PushButton ButtonAccettaSubmit`: bottone per accettare la richiesta di chiamata.

`@UiField PushButton ButtonRifiutaSubmit`: bottone per rifiutare la richiesta di chiamata.

`@UiField FormPanel FormInfoChiamata`: pannello contenente `LabelInfoUtente`, `LabelDurata`, `LabelLatenza`, `LabelByte`, `LabelPack`, `LabelLostPack`, `PanelVote` e `ButtonChiudiChiamataSubmit`.

`@UiField InlineLabel LabelInfoUtente`: label che contiene il nome dell'utente coinvolto nella comunicazione.

`@UiField InlineLabel LabelDurata`: label che contiene la durata della comunicazione.

`@UiField InlineLabel LabelLatenza`: label che contiene la latenza della comunicazione.

`@UiField InlineLabel LabelByte`: label che contiene il numero di byte inviati durante la comunicazione.

`@UiField InlineLabel LabelPack`: label che contiene il numero di pacchetti inviati durante la comunicazione.

`@UiField InlineLabel LabelLostPack`: label che contiene il numero di pacchetti persi durante la comunicazione.

`@UiField HorizontalPanel PanelVote`: pannello orizzontale contenente i `RadioButton` per ricavare l'informazione del giudizio dell'utente riguardo alla comunicazione effettuata.

`@UiField PushButton ButtonChiudiChiamataSubmit`: bottone per l'invio della richiesta di chiusura della comunicazione.

`@UiField CaptionPanel CaptionChiamataInCorso`: pannello contenente `LabelTimer`.

`@UiField InlineLabel LabelTimer`: label che indica lo stato di attesa del mittente di una richiesta di chiamata attraverso un conto alla rovescia.

Metodi:
`+ CommunicationView(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocket);`
Costruttore: inizializza `communicationLogic` con i valori ricevuti come parametri, imposta inoltre `webSocket`, `makeCall`, `callManage` e `mediaManage` con il valore del riferimento `communicationLogic`. Imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$ e imposta video a false e count a 0. Imposta invisibili tutte le `LabelError` e i disabilita i bottoni `Chiama` e `Lascia Messaggio`. Inoltre crea gli oggetti `ListUser` e `ListMessage`.

`+ void loadViewCommunication();`
Visualizza la $GUI_{|g|}$ di comunicazione e richiama i metodi `graphicInitialCharge()` e attraverso il riferimento `communicationLogic` richiama `setUserList()` che crea una richiesta per ottenere la lista degli utenti.

`+ void removeViewCommunication();`
Nasconde la $GUI_{|g|}$ di comunicazione.

`+ void setUsernameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'utente autenticato passato attraverso la stringa `name`.

`+ void errorView();`
Rende visibili le `LabelError`

`+ void graphicInitialCharge();`
Nel caso in cui più richieste di comunicazione siano presenti tale metodo non aggiorna la $GUI_{|g|}$, in caso contrario imposta la $GUI_{|g|}$ di comunicazione allo stato iniziale. Tale stato è il seguente:

- `count`: 0
- `timer`: reset
- `TabBar`: visibile
- `LinkListaUtenti`: selezionato
- `FormListaUtenti`: visibile

- `tutte le LabelError`: valori iniziali e invisibili
- `FormRicercaUtente`: invisibile
- `FormRicercaIP`: invisibile
- `CaptionChiamataInCorso`: invisibile
- `FormRicezioneChiamata`: invisibile
- `FormInfoChiamata`: invisibile
- `DivVideo`: HTML immagine di attesa

`+ void setVideoLocal(String localUrl);`
Imposta il tag `DivVideoLocal` con il codice $\text{HTML}_{|g|}$ per il video con l'indirizzo passato attraverso la stringa `localUrl`. Inoltre rende invisibile `FormActivateLocalVideo`, attiva i bottoni `Chiama` (nel caso della lista utenti solo di quelli online) e imposta video a `true`.

`+ void waitView(String nameInterlocutor);`
Rende invisibile `TabBar`, `FormListaUtenti`, `FormRicercaUtente`, `FormRicercaIP` e `FormMessaggi`. Rende visibile `CaptionChiamataInCorso` e fa partire il conto alla rovescia per l'attesa della risposta. Imposta inoltre il nome del destinatario con la stringa `utente` e incrementa `count`.

`- void callRefuse(String nameInterlocutor);`
Se il numero di chiamate in attesa è diverso da 1 allora viene solo decrementato `count`, altrimenti viene richiamato il metodo `graphicInitialCharge()` e richiamato attraverso il riferimento `callManage` il metodo `callStart(false, nameInterlocutor)` che dichiara che la negoziazione verso l'utente contenuto nel parametro `nameInterlocutor` è fallita.

`+ void externalCall(String nameInterlocutor);`
Viene incrementato `count` (il numero di chiamate in attesa). Se il video è attivo e `count` vale 1 allora `TabBar`, `FormListaUtenti`, `FormRicercaUtente` e `FormRicercaIP` vengono resi invisibili, e viene visualizzato `FormRicezioneChiamata` impostando ad `nameInterlocutor` il contenuto di `LabelNomeUtente`. Nel caso in cui il video non sia attivo o il numero di chiamate in attesa sia diverso da 1 allora viene visualizzato un messaggio di errore.

`+ void callActive(String remoteUrl, String nameInterlocutor);`
Blocca e reimposta il timer. Inoltre rende invisibili `FormRicezioneChiamata` e `CaptionChiamataInCorso`. Rende visibile `FormInfoChiamata` impostando `LabelInfoUtente` con il parametro `nameInterlocutor`. Inoltre, imposta il `DivVideo` con l'indirizzo del video remoto dell'utente remoto contenuto nel parametro `remoteUrl`.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Imposta le Label di `FormInfoChiamata` con i parametri passati nel vettore

`dataCall`.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ della comunicazione. Richiama il metodo `close()` attraverso il riferimento `callManage` nel caso in cui ci sia una chiamata attiva e richiama il metodo `logoutUser()` attraverso il riferimento `dataUserLogic`.

`- void contactList();`
Metodo che crea l'oggetto `cellList` e gli abbina il metodo per la selezione di un singolo elemento all'interno della lista.

`- void messageList();`
Metodo che crea l'oggetto `cellListMessage` e gli abbina il metodo per la selezione di un singolo elemento all'interno della lista.

`+ void setList(Vector<String> listUser);`
Metodo che popola con il vettore `listUser` l'oggetto `cellList`.

`+ void setListMessage(Vector<String> listMessage);`
Metodo che popola con il vettore `listMessage` l'oggetto `cellListMessage` e aggiorna la grafica evidenziando il caso in cui siano arrivati nuovi messaggi.

`+ void setLabelIP(String ipAddress);`
Imposta l'etichetta `LabelIP` con l'indirizzo $IP_{|g|}$ dell'utente autenticato passato attraverso la stringa `ipAddress`.

`+ void terminate();`
Metodo che imposta `count` a 0 e viene richiamato, attraverso il riferimento `callManage`, il metodo `close(value)` che richiede la terminazione della comunicazione inviando il voto dell'utente. Richiama, inoltre, il metodo `graphicInitialCharge()`.

Eventi:
`@UiHandler void onLinkDatiUtenteClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkDataUser` viene richiamato il metodo `removeViewCommunication()` e attraverso il riferimento `communicationLogic` richiama `loadViewDataUser()`. Se sono presenti chiamate attive richiama, attraverso il riferimento `callManage`, il metodo `close()` e pone `count` a 0.

`@UiHandler void onLinkLogoutClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkLogout` viene richiamato il metodo `removeViewCommunication()`. Inoltre utilizza il riferimento `communicationLogic` per effettuare il logout e richiamare la GUI di autenticazione attraverso il metodo `logoutUser()`. Se sono presenti chiamate

attive richiama, attraverso il riferimento `callManage`, il metodo `close()` e pone count a 0.

`@UiHandler void onAttivaVideoSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `AttivaVideoSubmit` viene richiamato, attraverso il riferimento `mediaManage`, il metodo `activeVideoLocal()`.

`@UiHandler void onLinkListaUtentiClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkListaUtenti` viene resa visibile `FormListaUtenti` e vengono rese invisibili `FormRicercaUtente`, `FormRicercaIP` e `FormMessaggi`.

`@UiHandler void onLinkRicercaUtenteClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkRicercaUtente` viene resa visibile la `FormRicercaUtente` e vengono rese invisibili `FormListaUtenti`, `FormRicercaIP` e `FormMessaggi`.

`@UiHandler void onLinkRicercaIPClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkRicercaIP` viene resa visibile la `FormRicercaIP` e vengono rese invisibili `FormRicercaUtente`, `FormListaUtenti` e `FormMessaggi`.

`@UiHandler void onLinkMessaggiClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkMessaggi` viene resa visibile la `FormMessaggi` e vengono rese invisibili `FormRicercaUtente`, `FormListaUtenti` e `FormRicercaIP`.

`@UiHandler void onBoxChiamaLUSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `BoxChiamaLUSubmit`, solo se è stato selezionato un utente online, viene inviata una richiesta di comunicazione all'utente selezionato attraverso il riferimento `makeCall`, il quale richiama il metodo `searchUser(String type, String nameInterlocutor)`.

`@UiHandler void onButtonSegreteriaClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonSegreteria`, solo se è stato selezionato un utente, rende visibile la finestra di dialogo che permette di creare un messaggio da inviare all'utente selezionato.

`@UiHandler void onButtonRicercaChiamaSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `BoxRicercaChiamaSubmit` viene inviata una richiesta di comunicazione all'utente selezionato attraverso il riferimento `makeCall` che richiama il metodo `searchUser(String tipo, String utente)`.

`@UiHandler void onButtonRicercaIPChiamaSubmitClick(ClickEvent event);`

All'evento `Click` dell'oggetto `BoxRicercaIPChiamaSubmit` viene inviata una richiesta di comunicazione all'utente selezionato attraverso il riferimento `makeCall`, il quale richiama il metodo `searchUser(String tipo, String utente)`.

`@UiHandler void onButtonEliminaTuttiClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonEliminaTutti`, gli elementi presenti all'interno della `listMessage` e attraverso il riferimento a `callManage` richiama il metodo `eraseMessageAll()`.

`@UiHandler void onButtonAccettaSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonAccettaSubmit` vengono resi invisibili `FormRicezioneChiamata` e `CaptionChiamataInCorso`. Inoltre, viene chiamato, attraverso il metodo `callManage`, il metodo `callStart(true, utente)` che permette di instaurare la comunicazione.

`@UiHandler void onButtonRifiutaSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonRifiutaSubmit` chiama il metodo `graphicInitialCharge()`. Inoltre, viene chiamato, attraverso il metodo `callManage`, il metodo `callStart(false, utente)` che permette di rifiutare la comunicazione.

`@UiHandler void onButtonChiudiChiamataSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonChiudiChiamataSubmit`, `count` viene impostato a 0 e viene richiamato, attraverso il riferimento `callManage`, il metodo `close(value)` che richiede la terminazione della comunicazione inviando il voto dell'utente. Richiama, inoltre, il metodo `graphicInitialCharge()`.

`@UiHandler void onWindowClosing(Window.ClosingEvent event);`
Alla chiusura della finestra vengono richiamati, attraverso i riferimenti `callManage` e `communicationLogic`, ripettivamente i metodi `close(0)` e `logoutUser()`. Il metodo `close(0)` viene richiamato solo nel caso in cui `count` sia maggiore di 0.

3.2.3 LogUser

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di autenticazione;
- gestire gli eventi che l'utente o il sistema possono innescare inoltrando le richieste all'interfaccia `mytalk.client.presenter.user.logicUser.ILogUserLogic`;
- gestire i link presenti nella pagina.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.ILogUser`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.LogUserLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.ILogUserLogic`.

Attributi:
`- ILogUserLogic logUserLogic;`
Riferimento alla classe `LogUserLogic`.

Oggetti:
`@UiField HTMLPanel MyDivLogin;`
Pannello contente `FormLogUser` e `LinkRegistrati`.

`@UiField FormPanel FormLogUser;`
Form contenente `BoxUtente`, `BoxPassword`, `LabelError` e `LogUserSubmit`.

`@UiField TextBox BoxUtente;`
Campo per l'inserimento del nome utente.

`@UiField TextBox BoxPassword;`
Campo per l'inserimento della password.

`@UiField InlineLabel LabelError;`
Label che in presenza di errore segnala il tipo di errore.

`@UiField PushButton LogUserSubmit;`
Bottone per l'invio della richiesta controllo delle credenziali di autenticazione.

`@UiField InlineHyperlink LinkRegistrati;`
Link per passare alla grafica della registrazione.

Metodi:
`+ LogUser(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocket);`
Costruttore: inizializza `logUserLogic` con i valori ricevuti, imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$. Imposta come invisibile `LabelError`.

`+ void loadViewLogUser();`
Visualizza la $GUI_{|g|}$ di autenticazione e seleziona le impostazioni di $default_{|g|}$: `BoxUtente` e `BoxPassword` vuoti e `LabelError` invisibile.

`+ void removeViewLogUser();`
Nasconde la $GUI_{|g|}$ di login.

`+ void errorView(String error);`
Imposta e rende visibile `LabelError` inserendo il contenuto della stringa `error`.

Eventi:
`@UiHandler void onLinkRegistratiClick(ClickEvent event);`
All'evento `Click` dell'utente richiama il metodo `removeViewLogUser()` e, attraverso il riferimento `logUserLogic`, richiama il metodo `loadViewRegister()`.

`@UiHandler void onLogUserSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LogUserSubmit` i dati inseriti all'interno dei campi `BoxUtente` e `BoxPassword` vengono inseriti in un vettore e inviati attraverso il riferimento `logUserLogic` al metodo `validateData(Vector<String>)` per effettuare il controllo dei dati di autenticazione.

3.2.4 MakeCall

Funzione:
La classe ha il compito di inoltrare al Presenter le richieste per la gestione della negoziazione di una comunicazione.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IMakeCall`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic`.

Attributi:
`- ICommunicationLogic communicationLogic`: riferimento alla classe `CommunicationLogic`.

Metodi:
`+ MakeCall(IComunicationLogic communicationLogic);`

Costruttore: inizializza `communicationLogic` con il valore ricevuto come parametro.

`+ void searchUser(String type, String key);`
Invia una richiesta al Presenter per cercare l'utente desiderato e controllarne la disponibilità. L'attributo `type` indica in che chiave viene ricercato l'utente e `key` è la stringa per il confronto con il database$_{|g|}$.

3.2.5 MediaManage

Funzione:
La classe ha il compito di inoltrare al Presenter le richieste per la gestione dei media per la comunicazione.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IMediaManage`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic`.

Attributi:
`- ICommunicationLogic communicationLogic`: riferimento alla classe `CommunicationLogic`.

Metodi:
`+ MediaManage(IComunicationLogic comunicationLogic);`
Costruttore: inizializza `CommunicationLogic` con il valore ricevuto come parametro.

`+ void activeVideoLocal();`
Invia una richiesta al Presenter attraverso il metodo `getLocalUrl()` per recuperare l'indirizzo del video locale.

3.2.6 ModifyDataUser

Funzione:
La classe ha il compito di inoltrare al Presenter le richieste per la modifica dei dati utente.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IModifyDataUser`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.DataUserLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.IDataUserLogic`.

Attributi:
`- IDataUserLogic dataUserLogic;`
Riferimento alla classe `DataUserLogic`.

Metodi:
`+ ModifyDataUser(IDataUserLogic dataUserLogic);`
Costruttore: inizializza `dataUserLogic` con il valore ricevuto come parametro.

`+ void sendRequest(Vector <String> data);`
Invia una richiesta, attraverso il riferimento `dataUserLogic`, al metodo `checkNewData(Vector<String>)` che controlla che i nuovi dati dell'utente siano scritti in modo corretto e, quindi, li salva o ritorna un messaggio di errore.

3.2.7 PageUserView

Funzione:
La classe ha il compito di smistare le richieste di aggiornamento della View provenienti dal Presenter.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IPageUserView`.

Usa le classi:

`mytalk.client.view.user.LogUserView`;

`mytalk.client.view.user.RegisterView`;

`mytalk.client.view.user.UserDataView`;

`mytalk.client.view.user.CommunicationView`;

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`.

Tramite le interfacce:

`mytalk.client.iView.iUser.ILogUserView`;

`mytalk.client.iView.iUser.IRegisterView`;

`mytalk.client.iView.iUser.IUserDataView`;

`mytalk.client.iView.iUser.ICommunicationView`;

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic`.

Attributi:
- `ILogUser logUser`: riferimento alla classe `LogUser`.

- `IRegister register`: riferimento alla classe `Register`.

- `IUserDataView userDataView`: riferimento alla classe `UserDataView`.

- `ICommunicationView communication`: riferimento alla classe `CommunicationView`.

Metodi:
`+ PageUserView()`
Costruttore: inizializza, carica nel pannello di root e imposta la visibilità degli oggetti:

- `logUser`;
- `register`;
- `userDataView`;
- `communication`.

Imposta poi un listener che attende l'evento di chiusura della pagina e che che avvisa l'utente e in caso richiama i metodi di chiusura degli oggetti `userDataView` e `communication`.

`+ void updateViewLogUser(boolean update, String message);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: richiama il metodo `errorView(message)` attraverso il riferimento `logUser` per visualizzare gli errori di autenticazione;
- se `true`: richiama il metodo `removeViewLogUser()` attraverso il riferimento `logUser` e successivamente richiama il metodo `loadViewCommunication()` attraverso il riferimento `communication`.

`+ void notifyExternalCall(String nameInterlocutor);`
Richiama il metodo `externalCall(nameInterlocutor)` attraverso il riferimento `communication` per notificare una richiesta di comunicazione in arrivo.

`+ void notifyRefuseCall(String nameInterlocutor);`
Richiama il metodo `graphicInitialCharge()` attraverso il riferimento `communication` per notificare il rifiuto di una richiesta di comunicazione.

`+ void setUserList(Vector<String> listUser);`
Richiama il metodo `loadListBox(listUser)` attraverso il riferimento `communication` per impostare la lista utenti della comunicazione.

`+ void loadViewRegister();`
Richiama il metodo `loadViewRegister()` attraverso il riferimento `register` di approvazione del file per visualizzare la registrazione.

`+ void loadViewLogUser();`
Richiama il metodo `loadViewLogUser()` attraverso il riferimento `logUser` per visualizzare l'autenticazione.

`+ void updateViewRegister(boolean update, Vector<String> messages);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: richiama il metodo `errorView(messages)` attraverso il riferimento `register` per visualizzare gli errori di registrazione.
- se `true`: richiama il metodo `removeViewRegister()` attraverso il riferimento `register` e successivamente richiama il metodo `loadViewLogUser()` attraverso il riferimento `logUser`.

`+ void loadViewUserData();`
Richiama il metodo `loadViewUserData()` attraverso il riferimento `userDataView` per visualizzare la gestione dai dell'utente.

`+ void setLocalVideo(String localUrl);`
Richiama il metodo `setLocalVideo(localUrl)` attraverso il riferimento `communication` per impostare il video locale.

`+ void setUsernameLabel(String name);`
Richiama il metodo `setUsenameLabel(name)` attraverso i riferimenti `userDataView` e `communication` per impostare il nome dell'utente autenticato.

`+ void loadViewCommunication();`
Richiama il metodo `loadViewCommunication()` attraverso il riferimento `communication` per visualizzare la $GUI_{|g|}$ di comunicazione.

`+ void updateViewDataUser(boolean update, Vector<String> messages);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: richiama il metodo `errorView(messages)` attraverso il riferimento `updateUserView` per visualizzare gli errori di modifica dei dati;

- se `true`: richiama il metodo `loadViewUserData()` attraverso il riferimento `userDataView`.

`+ void setUserDataLabel(Vector<String> userData);`
Richiama il metodo `setLabel(userData)` attraverso il riferimento `userDataView` per impostare le etichette che visualizzano i dati dell'utente.

`+ void callActive(String remoteUrl, String nameInterlocutor);`
Richiama il metodo `callActive(remoteUrl, nameInterlocutor)` attraverso il riferimento `communication` per notificare l'attivazione di una conversazione.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Richiama il metodo `updateFormInfoChiamata(dataCall)` attraverso il riferimento `communication` per impostare le etichette che visualizzano le statistiche della comunicazione.

`+ void updatePanelSearch(boolean update, String nameInterlocutor);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: richiama il metodo `errorView()` attraverso il riferimento `communication`;
- se `true`: richiama il metodo `waitView()` che visualizza il pannello di attesa di risposta da parte dell'utente in fase di negoziazione.

`+ void setIPLabel(String ipAddress);`
Richiama il metodo `setIPLabel(ipAddress)` attraverso il riferimento `communication` per impostare l'etichetta che visualizza il proprio indirizzo $IP_{|g|}$.

`+ void terminate();`
Richiama il metodo `terminate()` attraverso il riferimento `communication` per la conclusione di una comunicazione.

`+ void setSegreteria(Vector<String> messagges);`
Richiama il metodo `setSegreteria(messages)` attraverso il riferimento `communication` per inviare alla $GUI_{|g|}$ di comunicazione la lista dei messaggi presenti in segreteria.

3.2.8 Register

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di registrazione;

- gestire gli eventi che l'utente o il sistema possono innescare inoltrando le richieste all'interfaccia `IRegisterLogic`;
- gestire i link presenti nella pagina.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IRegister`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.RegisterLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.IRegisterLogic`.

Attributi:
`- IRegisterLogic registerLogic;`
Riferimento alla classe `RegisterLogic`.

Oggetti:
`@UiField HTMLPanel MyDiv;`
Pannello contente `FormRegister` e `LinkLogin`.

`@UiField FormPanel FormRegister;`
Form contenente `GridRegistrati`.

`@UiField Grid GridRegistrati;`
Griglia contenente `BoxEMail`, `LabelErrorBoxEMail`, `BoxNome`, `LabelErrorBoxNome`, `BoxCognome`, `LabelErrorBoxCognome`, `BoxPassword`, `LabelErrorBoxPassword`, `BoxRipPass`, `LabelErrorBoxRipPass`, `BoxAzienda`, `LabelErrorBoxAzienda`, `BoxTelefono`, `LabelErrorBoxTelefono` e `ButtonRegistratiSubmit`.

`@UiField TextBox BoxEMail;`
Campo per l'inserimento dell'e-mail.

`@UiField InlineLabel LabelErrorBoxEMail;`
Label che contiene l'errore relativo all'e-mail errata.

`@UiField TextBox BoxNome;`
Campo per l'inserimento del nome.

`@UiField InlineLabel LabelErrorBoxNome;`
Label che contiene l'errore relativo al nome errato.

`@UiField TextBox BoxCognome;`
Campo per l'inserimento del cognome.

`@UiField InlineLabel LabelErrorBoxCognome;`
Label che contiene l'errore relativo al cognome errato.

`@UiField TextBox BoxPassword;`
Campo per l'inserimento della password.

`@UiField InlineLabel LabelErrorBoxPassword;`
Label che contiene l'errore relativo alla password errata.

`@UiField TextBox BoxRipPass;`
Campo per l'inserimento della password di conferma.

`@UiField InlineLabel LabelErrorBoxRipPass;`
Label che contiene l'errore relativo alla non uguaglianza delle due password inserite.

`@UiField TextBox BoxAzienda;`
Campo per l'inserimento dell'azienda.

`@UiField InlineLabel LabelErrorBoxAzienda;`
Label che contiene l'errore relativo al nome dell'azienda errato.

`@UiField TextBox BoxTelefono;`
Campo per l'inserimento del numero di telefono.

`@UiField InlineLabel LabelErrorBoxTelefono;`
Label che contiene l'errore relativo al numero di telefono errato.

`@UiField InlineHyperlink LinkLogin;`
Link per passare alla $GUI_{|g|}$ di autenticazione.

`@UiField PushButton ButtonRegistratiSubmit;`
Bottone per l'invio della richiesta di controllo delle credenziali relative alla registrazione.

Metodi:
`+ Register(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocket);`

Costruttore: inizializza `registerLogic` con i valori ricevuti, imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$. Imposta invisibili le `labelError`.

`+ void loadViewRegister();`
Visualizza la $GUI_{|g|}$ di registrazione.

`+ void removeViewRegister();`
Nasconde la $GUI_{|g|}$ di registrazione.

`+ void errorView(Vector<String> messages);`
Copia le stringhe di errore contenute nel vettore `messages` nei corrispondenti campi `LabelError`, che successivamente visualizza.

Eventi:
`@UiHandler void onLinkLoginClick(ClickEvent event);`
All'evento `Click` dell'utente richiama il metodo `removeViewRegister()` e, attraverso il riferimento `registerLogic`, richiama il metodo `loadViewLogUser()`.

`@UiHandler void onButtonRegistratiSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonRegistratiSubmit` i dati inseriti all'interno dei campi `BoxEMail`, `BoxNome`, `BoxCognome`, `BoxPassword`, `BoxRipPass`, `BoxAzienda` e `BoxTelefono` vengono inseriti in un vettore e inviati attraverso il riferimento `registerLogic` al metodo `validateData(Vector<String>)`, il quale effettua il controllo dei dati di registrazione.

3.2.9 ShowDataUser

Funzione:
La classe ha il compito di inoltrare al Presenter le richieste per la visualizzazione dei dati utente.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IShowDataUser`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.DataUserLogic`;

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.IDataUserLogic`.

Attributi:
- `IDataUserLogic dataUserLogic;`
Riferimento alla classe `DataUserLogic`.

Metodi:
`+ ShowDataUser(IDataUserLogic dataUserLogic);`
Costruttore: inizializza `dataUserLogic` con il valore ricevuto come parametro.

`+ void dataRequest();`
Invia una richiesta, attraverso il riferimento `dataUserLogic`, al metodo `getDataUser()` per reperire i dati dell'utente.

3.2.10 UserDataView

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di gestione dei dati dell'utente autenticato;
- gestire gli eventi che l'utente o il sistema possono innescare inoltrando le richieste di visualizzazione all'interfaccia `IShowDataUser` e le richieste di modifica dei dati all'interfaccia `IModifyDataUser`;
- gestire i link presenti nella pagina.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iUser.IUserDataView`.

Usa le classi:

`mytalk.client.view.user.ShowDataUser`;

`mytalk.client.view.user.ModifyDataUser`;

`mytalk.client.presenter.user.logicUser.DataUserLogic`.

Tramite le interfacce:

`mytalk.client.iView.iUser.IShowDataUser`;

`mytalk.client.iView.iUser.IModifyDataUser`;

`mytalk.client.iPresenter.iUser.LogicUser.IDataUserLogic`.

Attributi:
- `IDataUserLogic dataUserLogic;`
Riferimento alla classe `DataUserLogic`.

- `IShowDataUser showData;`
Riferimento alla classe `ShowDataUser`.

- `IModifyDataUser modifyData;`
Riferimento alla classe `ModifyDataUser`.

- `boolean checkModify;`
Indica se l'utente desidera modificare la password.

- `String attPassword;`
Password attuale.

Oggetti:
`@UiField HTMLPanel MyDiv;`
Pannello contente `FormShowData`, `FormModifyData`, `LabelUser`, `LinkCommunication` e `LinkLogout`.

`@UiField InlineHyperlink LinkCommunication;`
Link per passare alla $GUI_{|g|}$ della comunicazione.

`@UiField InlineHyperlink LinkLogout;`
Link per passare alla $GUI_{|g|}$ dell'autenticazione effettuando il logout.

`@UiField InlineLabel LabelUser;`
Label che visualizza lo username dell'utente autenticato.

`@UiField CaptionPanel DatiUtente;`
Pannello contenente la `FormShowData`.

`@UiField FormPanel FormShowData;`
Form contenente `LabelNome`, `LabelCognome`, `LabelEMail`, `LabelAzienda`, `LabelTelefono` e `ButtonModificaSubmit`.

`@UiField InlineLabel LabelNome;`
Label contenente il nome dell'utente.

`@UiField InlineLabel LabelCognome;`
Label contenente il cognome dell'utente.

`@UiField InlineLabel LabelEmail;`
Label contenente l'e-mail dell'utente.

`@UiField InlineLabel LabelAzienda;`
Label contenente il nome dell'azienda dell'utente.

`@UiField InlineLabel LabelTelefono;`
Label contenente il numero di telefono dell'utente.

`@UiField PushButton ButtonModificaSubmit;`
Bottone per passare al `CaptionPanel ModificaDati`.

`@UiField CaptionPanel ModificaDati;`
Pannello contenente `FormModifyData`.

`@UiField FormPanel FormModifyData;`
Form contenente `BoxNome`, `BoxCognome`, `BoxEMail`, `BoxAzienda`, `BoxTelefono`, `BoxOldPass`, `BoxModifyCheck`, `BoxNewPass`, `BoxConfPass`, `LabelErrorNome`, `LabelErrorCognome`, `LabelEmailError`, `LabelAziendaError`, `LabelTelefonoError`, `LabelOldError`, `LabelNewError`, `LabelConfPassError`, `ButtonConfermaSubmit` e `ButtonIndietroSubmit`.

`@UiField TextBox BoxNome;`
Campo per l'inserimento del nome.

`@UiField InlineLabel LabelErrorNome;`
Label che contiene l'errore relativo al nome errato.

`@UiField TextBox BoxCognome;`
Campo per l'inserimento del cognome.

`@UiField InlineLabel LabelErrorCognome;`
Label che contiene l'errore relativo al cognome errato.

`@UiField TextBox BoxEmail;`
Campo per l'inserimento dell'e-mail.

`@UiField InlineLabel LabelErrorEmail;`
Label che contiene l'errore relativo all'e-mail errata.

`@UiField TextBox BoxNome;`
Campo per l'inserimento del nome.

`@UiField InlineLabel LabelErrorNome;`
Label che contiene l'errore relativo al nome errato.

`@UiField TextBox BoxAzienda;`
Campo per l'inserimento del nome dell'azienda.

`@UiField InlineLabel LabelErrorAzienda;`
Label che contiene l'errore relativo al nome dell'azienda.

`@UiField TextBox BoxTelefono;`
Campo per l'inserimento del numero di telefono.

`@UiField InlineLabel LabelErrorTelefono;`
Label che contiene l'errore relativo al numero di telefono errato.

`@UiField TextBox BoxOldPass;`
Campo per l'inserimento della password.

`@UiField InlineLabel LabelOldError;`
Label che contiene l'errore relativo alla password attuale errata.

`@UiField SimpleCheckBox BoxModifyCheck;`
Controllo per attivare la modifica della password.

`@UiField TextBox BoxNewPass;`
Campo per l'inserimento della nuova password.

`@UiField InlineLabel LabelNewError;`
Label che contiene l'errore relativo al formato della nuova password errato.

`@UiField TextBox BoxConfPass;`
Campo per l'inserimento della password di conferma.

`@UiField InlineLabel LabelConfPassError;`
Label che contiene l'errore relativo alla non uguaglianza delle due password inserite.

`@UiField PushButton ButtonConfermaSubmit;`
Bottone di invio dei dati al Presenter per il controllo del formato e della loro conformità.

`@UiField PushButton ButtonIndietroSubmit;`
Bottone per passare al `CaptionPanel DatiUtente`.

Metodi:
`+ UserDataView(IUpdateViewLogic updateViewLogic, IWebSocketUser`

`webSocket);`
Costruttore:

- inizializza `dataUserLogic` con i valori ricevuti come parametri;
- imposta `showData` e `modifyData` con il valore del riferimento `dataUserLogic`;
- imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$ e imposta `checkModify` a `false`;
- imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$ e imposta `checkModify` a `false`;
- imposta invisibili tutte le `LabelError` e anche `DatiUtente` e `ModificaDati`.

`+ void loadViewUserData();`
Visualizza la $GUI_{|g|}$ di gestione dei dati utente e richiama il metodo `graphicInitialCharge()`.

`+ void removeViewUserData();`
Nasconde la $GUI_{|g|}$ di gestione dei dati utente.

`+ void setUsernameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'utente autenticato passato attraverso la stringa `username`.

`+ void errorView(Vector<String> messages);`
Copia le stringhe di errore contenute nel vettore `messages` nei corrispondenti campi `LabelError`, che successivamente visualizza.

`+ void setLabel(Vector<String> data);`
Imposta e rende visibili le `Label` con i dati attuali dell'utente contenuti nel vettore `data`.

`- void graphicInitialCharge();`
Imposta la $GUI_{|g|}$ di gestione dei dati utente allo stato iniziale. Tale stato è il seguente:

- `DatiUtente`: visibile;
- `ModificaDati`: non visibile;
- tutte le `LabelError`: non visibili;

Inoltre, attraverso il riferimento `showData`, richiama il metodo `dataRequest()` che crea una richiesta per ottenere i dati dell'utente.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ di gestione dei dati. Richiama il metodo `logoutUser()` attraverso il riferimento `dataUserLogic`.

Eventi:
`@UiHandler void onLinkCommunicationClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkCommunication` viene richiamato il metodo `removeViewUserData()` e attraverso il riferimento `udateViewLogic` richiama `loadViewCommunication()`.

`@UiHandler void onLinkLogoutClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkLogout` viene richiamato il metodo `removeViewUserData()` il quale utilizza il riferimento `dataUserLogic` per effettuare il logout e richiama la $GUI_{|g|}$ dell'interfaccia di autenticazione attraverso il metodo `logoutUser()`.

`@UiHandler void onButtonModificaSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonModificaSubmit` è reso visibile il caption `ModificaDati` e reso invisibile il caption `DatiUtente`.

`@UiHandler void onButtonConfermaSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonConfermaSubmit` i dati inseriti all'interno dei campi `BoxEMail`, `BoxNome`, `BoxCognome`, `BoxOldPass`, `BoxNewPass`, `BoxConfPass`, `BoxAzienda`, `BoxTelefono` e `attPassword` vengono inseriti in un vettore e inviati attraverso il riferimento `modifyData` al metodo `sendRequest(Vector<String>)` che permette di inviare la richiesta, controllare i dati inviati e, nel caso siano corretti, di modificarli.

`@UiHandler void onButtonIndietroSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `ButtonIndietroSubmit` è reso visibile il caption `DatiUtente` e reso invisibile il caption `ModificaDati`.

`@UiHandler void onBoxModifyCheckClick(ClickEvent event);`
All'evento `Click` dell'oggetto `BoxModifyCheck` il campo `checkModify` viene impostato con il valore opposto a quello corrente (es. se era `true` diventa `false`) e `BoxNewPass`, `BoxConfPass` vengono resi invisibili o visibili a seconda del valore di `checkModify`.

3.3 Package mytalk.client.iView.iAdministrator

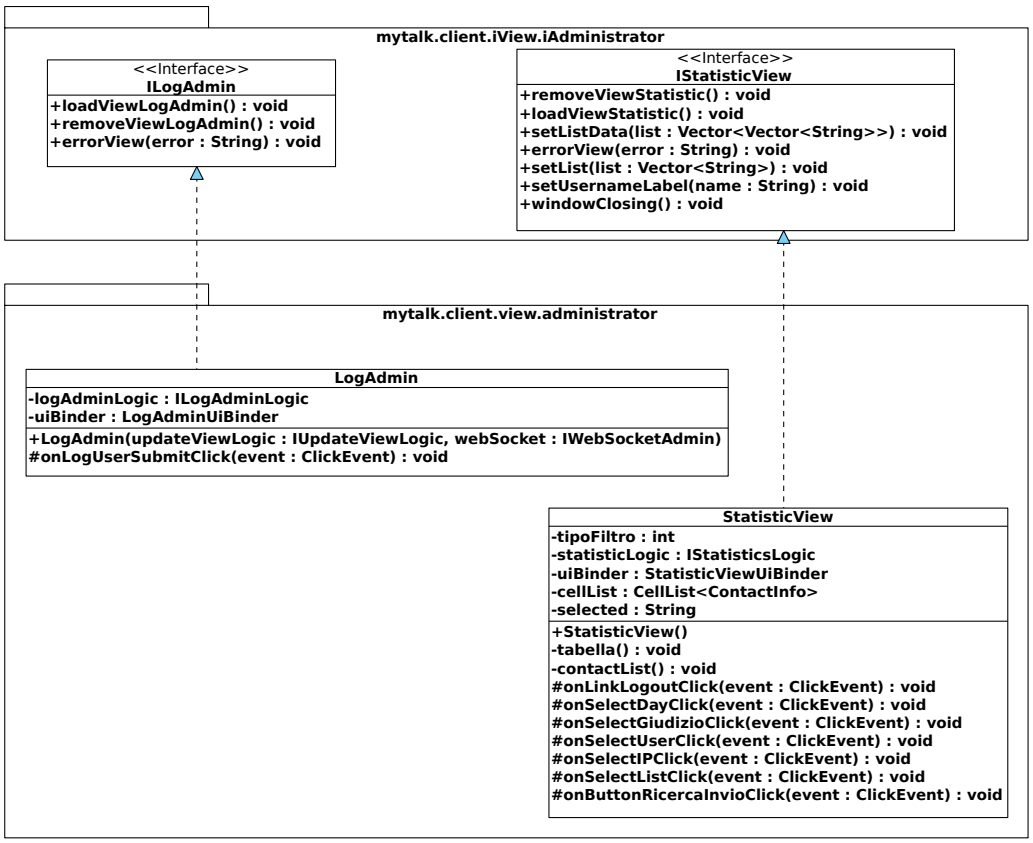

Figura 5: Diagramma delle classi dei package `mytalk.client.iView.iAdministrator` e `mytalk.client.view.administrator`; dettaglio delle classi `ILogAdmin`, `IStatisticView`, `LogAdmin` e `StatisticView`.

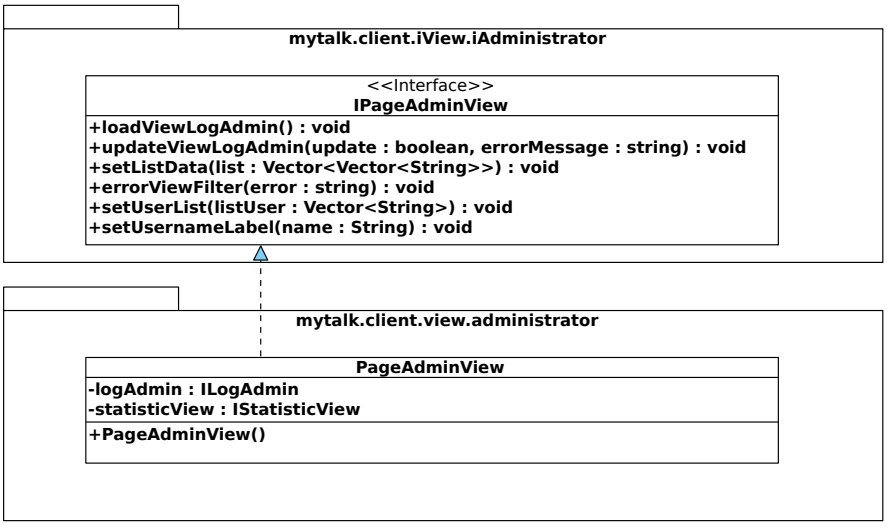

Figura 6: Diagramma delle classi dei package `mytalk.client.iView.iAdministrator` e `mytalk.client.view.administrator`; dettaglio delle classi `IPageAdminView` e `PageAdminView`.

3.3.1 ILogAdmin

Funzione:
Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ per l'autenticazione degli amministratori e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.administrator.LogAdmin`.

L'interfaccia è utilizzata da:

`mytalk.client.view.adminstrator.PageAdminView`.

Metodi:
`+ void loadViewLogAdmin();`
Visualizza la $GUI_{|g|}$ di autenticazione e seleziona le impostazioni di default$_{|g|}$, i campi vuoti e l'errore (non visibile).

`+ void removeViewLogAdmin();`
Nasconde la $GUI_{|g|}$ di autenticazione.

`+ void errorView(String error);`
Imposta e rende visibile il messaggio di errore.

3.3.2 IPageAdminView

Funzione:
Interfaccia che offre operazioni alle classi che compongono la $GUI_{|g|}$ e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.administrator.PageAdminView`.

L'interfaccia è utilizzata da:

`mytalk.client.MyTalkAdmin`;

`mytalk.client.presenter.administrator.logicAdmin.UpdateUserView`.

Metodi:
`+ void loadViewLogAdmin();`
Richiede la visualizzazione della $GUI_{|g|}$ di autenticazione dell'amministratore.

`+ void updateViewLogAdmin(boolean update, String errorMessage);`
Aggiorna la $GUI_{|g|}$ a seconda del valore di `update`:

- se `false`: aggiorna la $GUI_{|g|}$ di autenticazione visualizzando l'errore opportuno contenuto in `errorMessage`;
- se `true`: rende invisibile la $GUI_{|g|}$ di autenticazione e visualizza la $GUI_{|g|}$ di visualizzazione delle statistiche.

`+ void setListData(Vector<Vector<String>> list);`
Invia i dati delle statistiche alla tabella di visualizzazione.

`+ void errorViewFilter(String error);`
Notifica la presenza di errori nella chiave usata come filtro.

`+ void setUserList(Vector<String> listaUtenti);`
Imposta la lista degli utenti.

`+ void setUsernameLabel(String name);`
Invia il nome utente alla $GUI_{|g|}$ di visualizzazione delle statistiche.

3.3.3 IStatisticView

Funzione:
Offre operazioni alle classi che compongono la $GUI_{|g|}$ per la visualizzazione delle statistiche da parte dell'amministratore e interagiscono con essa.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.view.administrator.StatisticView`.

L'interfaccia è utilizzata da:

`mytalk.client.view.administrator.PageAdminView`.

Metodi:
`+ void removeViewStatistic();`
Nasconde la $GUI_{|g|}$ di visualizzazione statistiche.

`+ void loadViewStatistic();`
Visualizza la $GUI_{|g|}$ che mostra le statistiche.

`+ void setListData(Vector<Vector<String>> list);`
Inserisce nella tabella contenente le statistiche i dati contenuti nel parametro

`list`.

`+ void errorView(String error);`
Imposta e rende visibile il messaggio di errore.

`+ void setList(Vector<String> list);`
Inserisce gli utenti registrati contenuti nel vettore `list`.

`+ void setUsenameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'amministratore autenticato.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ di visualizzazione delle statistiche effettuando il logout.

3.4 Package mytalk.client.view.administrator

3.4.1 LogAdmin

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di autenticazione dell'amministratore;
- Data di approvazione del file gestire gli eventi che l'utente o il sistema possono innescare inoltrando le richieste all'interfaccia `ILogAdminLogic`.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iAdministrator.ILogAdmin`.

Usa le classi:

`mytalk.client.presenter.administrator.logicAdmin.LogAdminLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.ILogAdminLogic`.

Attributi:
- `ILogAdminLogic logAdminLogic`: riferimento alla classe `LogAdminLogic`.

Oggetti:
`@UiField HTMLPanel MyDivLogin`: pannello contente `FormLogUser`.

`@UiField FormPanel FormLogUser`: form contenente `BoxUtente`, `BoxPassword`, `LabelError` e `LogUserSubmit`.

`@UiField TextBox BoxUtente`: campo per l'inserimento del nome utente.

`@UiField TextBox BoxPassword`: campo per l'inserimento della password.

`@UiField InlineLabel LabelError`: label che in presenza di errore ne segnala il tipo.

`@UiField PushButton LogUserSubmit`: bottone per l'invio della richiesta di controllo delle credenziali di autenticazione dell'amministratore.

Metodi:
`+ LogAdmin(IUpdateViewLogic updateViewLogic, IWebSocketAdmin webSocket);`
Costruttore: inizializza `logAdminLogic` con i valori ricevuti come parametri, imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$ e imposta come invisibile `LabelError`.

`+ void loadViewLogAdmin();`
Visualizza la $GUI_{|g|}$ di autenticazione dell'amministratore e seleziona le impostazioni di $default_{|g|}$: `BoxUtente` e `BoxPassword` vuoti e `LabelError` invisibile.

`+ void removeViewLogAdmin();`
Nasconde la $GUI_{|g|}$ di login.

`+ void errorView(String error);`
Imposta e rende visibile `LabelError` inserendo il contenuto della stringa `error`.

Eventi:
`@UiHandler void onLogUserSubmitClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LogUserSubmit` i dati inseriti all'interno dei campi `BoxUtente` e `BoxPassword` vengono inseriti in un vettore e inviati attraverso il riferimento `logAdminLogic` al metodo `validateData(Vector<String>)` per effettuare il controllo dei dati di autenticazione.

3.4.2 PageAdminView

Funzione:
La classe ha il compito di smistare le richieste di aggiornamento della View da parte del Presenter.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iAdministrator.IPageAdminView`.

Usa le classi:

`mytalk.client.view.administrator.LogAdminView`;

`mytalk.client.view.administrator.StatisticView`;

`mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic`.

Tramite le interfacce:

`mytalk.client.iView.iAdministrator.ILogAdminView`;

`mytalk.client.iView.iAdministrator.IStatisticView`;

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IUpdateViewLogic`.

Attributi:
- `ILogAdmin logAdmin`: riferimento alla classe `LogAdmin`;

- `IStatisticView statisticView`: riferimento alla classe `StatisticView`.

Metodi:
`+ PageAdminView();`
Costruttore:

- inizializza, carica nel pannello di root e imposta la visibilità degli oggetti `logAdmin` e `statisticView`;
- imposta un listener che attende l'evento di chiusura della pagina e che in tal caso richiama i metodi di chiusura degli oggetti `statisticView`.

`+ void loadViewLogAdmin();`
Richiama il metodo `loadViewLogAdmin()` attraverso il riferimento `logAdmin` per visualizzare l'autenticazione dell'amministratore.

`+ void updateViewLogAdmin(boolean update, String errorMessage);`
Aggiorna la $GUI_{|g|}$ a seconda del valore `update`:

- se `false`: richiama il metodo `errorView(message)` attraverso il riferimento `logAdmin` per visualizzare gli errori di autenticazione;
- se `true`: richiama il metodo `removeViewLogAdmin()` attraverso il riferimento `logAdmin` e successivamente richiama il metodo `loadViewStatistic()` attraverso il riferimento `statisticView`.

`+ void setListData(Vector<Vector<String>> list);`
Invia i dati delle statistiche alla tabella di visualizzazione con il metodo `setListData(list)` attraverso il riferimento `statisticView`.

`+ void errorViewFilter(String error);`
Richiama il metodo `errorView(messages)` attraverso il riferimento `statisticView` per visualizzare gli errori di filtro di ricerca.

`+ void setUserList(Vector<String> listUser);`
Richiama il metodo `setList(listaUtenti)` attraverso il riferimento `statisticView` per impostare la lista utenti del filtro di ricerca per lista.

`+ void setUsernameLabel(String name);`
Richiama il metodo `setUsenameLabel(name)` attraverso il riferimento `statisticView`.

3.4.3 StatisticView

Funzione:
La classe ha il compito di:

- aggiornare opportunamente la $GUI_{|g|}$ di visualizzazione delle statistiche;
- gestire gli eventi che l'utente o il sistema possono innescare inoltrando le richieste all'interfaccia `IStatisticLogic`;

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iView.iAdministrator.IStatisticView`.

Usa le classi:

`mytalk.client.presenter.administrator.logicAdmin.StatisticLogic`.

Tramite le interfacce:

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IStatisticLogic`.

Attributi:
- `IStatisticLogic statisticLogic`: riferimento alla classe `StatisticLogic`;

- `int tipoFiltro`: identifica il tipo di indice usato:
  - 1: giorno;
  - 2: giudizio;
  - 3: nome utente;
  - 4: $IP_{|g|}$ (non attivo al momento);
  - 5: da lista.

- `CellList<ContactInfo> cellList`: identifica l'oggetto che contiene la lista degli utenti registrati.

- `String selected`: string che identifica l'utente attualmente selezionato.

Oggetti:
`@UiField HTMLPanel MyDiv`: pannello contente `DivStatisticVision` e `DivPanelControl`.

`@UiField HTMLPanel DivStatisticVision`: pannello contente `cellTable`.

`@UiField CellTable<Vector<Vector<String>>>`: tabella per la visualizzazione dei dati delle statistiche.

`@UiField HTMLPanel DivPanelControl`: pannello contenente `LinkLogout`, `LabelUser` e `FormFiltro`.

`@UiField InlineHyperlink LinkLogout`: link per passare alla $GUI_{|g|}$ dell'autenticazione dell'amministratore effettuando il logout.

`@UiField InlineLabel LabelUser:`: label che visualizza lo username dell'utente autenticato.

`@UiField FormPanel FormFiltro`: form contente `GridFiltro`.

`@UiField Grid GridFiltro`: griglia contenente `SelectDay`, `SelectGiudizio`, `SelectUser`, `SelectIP`, `SelectList`, `Aggiorna`, `ButtonRicercaInvio`, `LabelErrorRicerca`, `ListBoxUtenti`.

`@UiField PushButton SelectDay`: bottone per impostare il valore di `tipoFiltro` a 1 e rendere visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiField PushButton SelectGiudizio`: bottone per impostare il valore di `tipoFiltro` a 2 e rendere visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiField PushButton SelectUser`: bottone per impostare il valore di `tipoFiltro` a 3 e rendere visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiField PushButton SelectIP`: bottone per impostare il valore di `tipoFiltro` a 4 e rendere visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiField PushButton SelectList`: bottone per impostare il valore di `tipoFiltro` a 5 e rendere visibili `ListBoxUtenti` e `ButtonRicercaInvio`.

`@UiField PushButton Aggiorna`: bottone per inviare la richiesta di aggiornamento di `cellTable` e impostare invisibili `ListBoxUtenti`, `BoxRicerca` e `ButtonRicercaInvio`.

`@UiField PushButton ButtonRicercaInvio`: bottone per richiedere di filtrare i dati di `cellTable` a seconda del tipo di filtro e della chiave inserita in `BoxRicerca` o `ListBoxUtenti`.

`@UiField InlineLabel LabelErrorRicerca`: label che contiene l'errore relativo alla ricerca della chiave usata da `filtro`.

`@UiField ListBox ListBoxUtenti`: oggetto contenente la lista di tutti gli utenti registrati.

`@UiField TextBox BoxRicerca`: campo per l'inserimento della chiave da usare come filtro.

Metodi:
`+ StatisticView(IUpdateViewLogic updateViewLogic, IWebSocketAdmin webSocket);`
Costruttore: inizializza `communicationLogic` con i valori ricevuti come parametri, imposta inoltre `webSocket` con il valore del riferimento `statisticLogic`. Imposta gli attributi degli oggetti che compongono la $GUI_{|g|}$ e crea le colonne di `cellTable` attraverso il metodo `tabella()`.

`+ void removeViewStatistic();`
Nasconde la $GUI_{|g|}$ di visualizzazione statistiche.

`+ void loadViewStatistic();`

- visualizza la $GUI_{|g|}$ di comunicazione e imposta come attivi tutti i filtri;
- richiama il metodo `setUsernameLabel()` attraverso il riferimento `statisticLogic`;

- imposta visibile la tabella;
- richiama il metodo `getListData()` attraverso il riferimento `statisticLogic` per richiedere di popolare la tabella.

`+ void setListData(Vector<Vector<String>> list);`
Popola la tabella con le statistiche i dati contenuti nel parametro `list`.

`+ void errorView(String error);`
Rende visibile il messaggio di errore.

`+ void setList(Vector<String> list);`
Inserisce gli utenti registrati contenuti nel vettore `list` e li inserisce all'interno degli oggetti contenuti in `cellList`.

`+ void setUsenameLabel(String name);`
Imposta l'etichetta `LabelUser` con il nome dell'amministratore autenticato.

`+ void windowClosing();`
Gestisce l'evento di chiusura della finestra della $GUI_{|g|}$ di visualizzazione delle statistiche effettuando il logout.

`- void tabella();`
Metodo per la creazione delle colonne di `cellTable`.

`- void contactList();`
Metodo che crea l'oggetto `cellList` che contiene la lista delle celle contenenti le informazioni dei contatti.

Eventi:
`@UiHandler void onLinkLogoutClick(ClickEvent event);`
All'evento `Click` dell'oggetto `LinkLogout` viene richiamato il metodo `removeViewStatistic()`. Inoltre utilizza il riferimento `communicationLogic` per effettuare il logout e richiamare la $GUI_{|g|}$ di autenticazione attraverso il metodo `logoutUser()`.

`@UiHandler void onSelectDayClick(ClickEvent event);`
All'evento `Click` dell'oggetto `SelectDay` viene impostato `tipoFiltro` a 1. Viene disabilitato il bottone `SelectDay` e vengono resi visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiHandler void onSelectGiudizioClick(ClickEvent event);`
All'evento `Click` dell'oggetto `SelectGiudizio` viene impostato `tipoFiltro` a 2. Viene disabilitato il bottone `SelectGiudizio` e vengono resi visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiHandler void onSelectUserClick(ClickEvent event);`
All'evento `Click` dell'oggetto `SelectUser` viene impostato `tipoFiltro` a 3. Viene disabilitato il bottone `SelectUser` e vengono resi visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiHandler void onSelectIPClick(ClickEvent event);`
All'evento `Click` dell'oggetto `SelectIP` viene impostato `tipoFiltro` a 4. Viene disabilitato il bottone `SelectIP` e vengono resi visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiHandler void onSelectListClick(ClickEvent event);`
All'evento `Click` dell'oggetto `SelectList` viene impostato `tipoFiltro` a 5. Viene disabilitato il bottone `SelectList` e vengono resi visibili `BoxRicerca` e `ButtonRicercaInvio`.

`@UiHandler void onAggiornaClick(ClickEvent event);`
All'evento `Click` dell'oggetto `Aggiorna` vengono abilitati tutti i bottoni `Select` e vengono nascosti `ListBoxUtenti`, `BoxRicerca` e `ButtonRicercaInvio`. Inoltre viene richiamato il metodo `getListData()` attraverso il riferimento `statisticLogic`.

`@UiHandler void onButtonRicercaInvioClick(ClickEvent event)`
All'evento `Click` dell'oggetto `ButtonRicercaInvio` viene controllato il tipo di filtro e invocato il metodo `ricerca(chiave, tipoFiltro)` attraverso il riferimento `statisticLogic`.

4 Specifica della componente Presenter

La componente Presenter raccoglie tutti metodi per gestire le operazioni dell'applicazione e si suddivide in due categorie: Client$_{|g|}$ (vedi 4.1) e Server$_{|g|}$ (vedi 4.2).

4.1 Client

La sotto-componente Client$_{|g|}$ gestisce le richieste avanzate dalla componente View (vedi 3). A questo scopo dialoga con la sotto-componente Server$_{|g|}$ del Presenter (vedi 4.2) tramite la messaggi XML$_{|g|}$(vedi A).
È formata dalle classi:

`mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic` (sez. 4.1.1.1)

`mytalk.client.iPresenter.iUser.iLogicUser.ILogUserLogic` (sez. 4.1.1.3)

`mytalk.client.iPresenter.iUser.iLogicUser.IRegisterLogic` (sez. 4.1.1.4)

`mytalk.client.iPresenter.iUser.iLogicUser.IDataUserLogic` (sez. 4.1.1.2)

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic` (sez. 4.1.5.3)

`mytalk.client.presenter.user.logicUser.CommunicationLogic` (sez. 4.1.2.1)

`mytalk.client.presenter.user.logicUser.LogUserLogic` (sez. 4.1.2.3)

`mytalk.client.presenter.user.logicUser.RegisterLogic` (sez. 4.1.2.4)

`mytalk.client.presenter.user.logicUser.DataUserLogic` (sez. 4.1.2.2)

`mytalk.client.presenter.user.logicUser.UpdateViewLogic` (sez. 4.1.6.3)

`mytalk.client.presenter.iUser.iCommunication.IMediaStream` (sez. 4.1.3.1)

`mytalk.client.presenter.iUser.iCommunication.IPeerConnection` (sez. 4.1.3.2)

`mytalk.client.presenter.user.communication.MediaStream` (sez. 4.1.4.1)

`mytalk.client.presenter.user.communication.PeerConnection` (sez. 4.1.4.2)

`mytalk.client.presenter.user.communication.WebRTCAdaptee` (sez. 4.1.4.3)

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.ILogAdminLogic` (sez. 4.1.5.1)

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IStatisticLogic` (sez. 4.1.5.2)

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IUpdateViewLogic` (sez. 4.1.5.3)

`mytalk.client.presenter.administrator.logicAdmin.LogAdminLogic` (sez. 4.1.6.1)

`mytalk.client.presenter.administrator.logicAdmin.StatisticLogic` (sez. 4.1.6.2)

`mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic` (sez. 4.1.6.3)

`mytalk.client.presenter.user.logicUser.common.CommonFunctions` (sez. 4.1.9.1)

`mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser` (sez. 4.1.10.1)

`mytalk.client.presenter.user.serverComUser.WebSocketUser` (sez. 4.1.11.1)

4.1.1 Package mytalk.client.iPresenter.iUser.iLogicUser

4.1.1.1 ICommunicationLogic

Funzione:
Interfaccia che offre le operazioni alle classi che hanno il compito di comporre gli oggetti che serviranno per instaurare una nuova comunicazione, chiudere una comunicazione in atto, gestire le comunicazioni in ingresso e gestire l'accesso ai media locali e remoti.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.serverComUser.WebSocketUser`.

Metodi:
`+ void call(String utente);`
Effettua una chiamata verso l'utente identificato dalla stringa `utente`.

`+ void accept(String utente);`
Accetta la chiamata effettuata dall'utente identificato dalla stringa `utente`.

`+ void callEnter(String offer);`
Crea una descrizione della sessione locale rispetto alla descrizione di una sessione remota passata come parametro.

`+ void close(int voto);`
Invia un segnale che indica di terminare la chiamata. Il parametro passato come riferimento indica l'opinione data dall'utente al servizio.

`+ String getLocalURL();`
Restituisce l'$\text{URL}_{|g|}$ associato allo $\text{stream}_{|g|}$ locale.

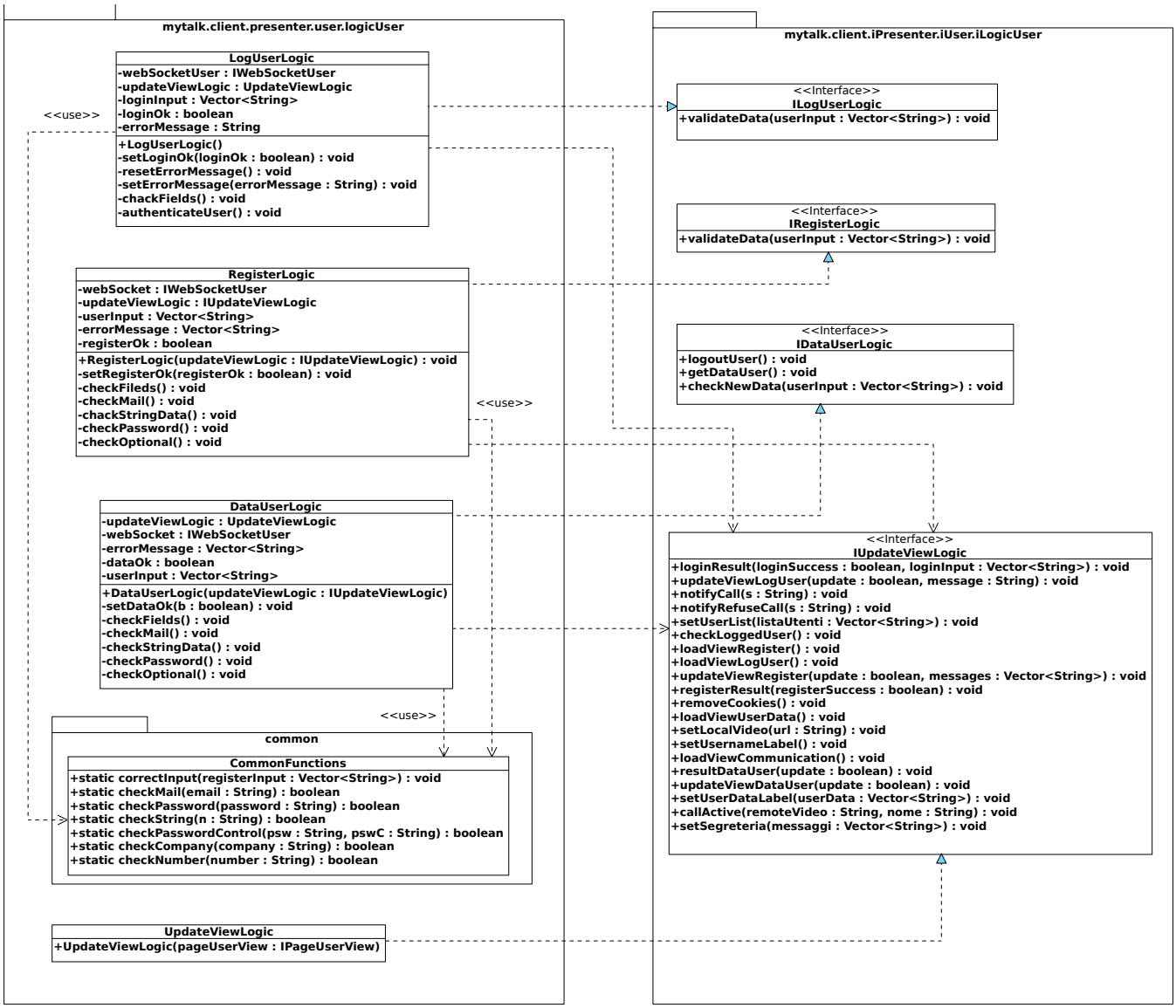

Figura 7: Diagramma delle classi dei package `mytalk.client.iPresenter.iUser.iLogicUser` e `mytalk.client.presenter.user.logicUser`; dettaglio delle classi `ILogUserLogic`, `IRegisterLogic`, `IDataUserLogic`, `IUpdateViewLogic`, `LogUserLogic`, `RegisterLogic`, `DataUserLogic`, `IUpdateViewLogic` e `mytalk.client.presenter.user.logicUser.common.CommonFunctions`.

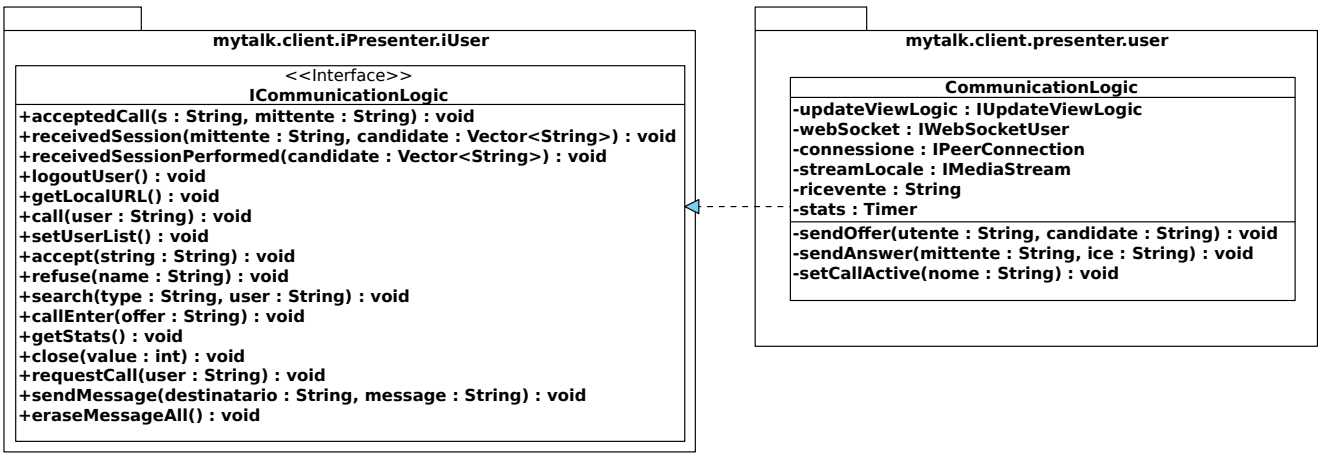

Figura 8: Diagramma delle classi dei package `mytalk.client.iPresenter.iUser` e `mytalk.client.presenter.user`; dettaglio delle classi `ICommunicationLogic` e `CommunicationLogic`.

`+ void getStats();`
Recupera le statistiche prelevate all'ultimo campionamento.

`+ void refuse(String utente);`
Rifiuta la chiamata dall'utente identificato dalla stringa `utente`.

`+ void acceptedCall(String answer, String utente);`
Notifica una chiamata accettata dall'utente `utente` e che quindi è stata precedentemente inviata.

`+ void receivedSessionPerformed(String mittente, Vector<String> iceCandiate`
Viene richiamato quando è stata effettuata una chiamata e questa è stata accettata dall'altro utente. Tale metodo permette di conoscere l'utente che ha inviato la descrizione e gli `IceCandidate` inviati in risposta.

`+ void receivedSession(String mittente, Vector<String> candidate);`
Permette di ricevere dei `candidate` e di gestirli in seguito alla ricezione di una comunicazione.

`+ void logoutUser();`
Permette all'utente di effettuare il logout.

`+ void setUserList();`
Invia al server$_{|g|}$ la richiesta della lista di utenti registrati al server$_{|g|}$.

`+ void search(String type, String user);`
Invia al server una richiesta riguardante l'esistenza di un utente registrato al

servizio.

`+ void setUserList();`
Richiede al server$_{|g|}$, tramite `WebSocket`, la lista degli utenti registrati al servizio.

`+ void setUsernameLabel();`
Imposta la label del nome dell'utente autenticato.

`+ void loadViewUserData();`
Richiama la GUI$_{|g|}$ relativa alla gestione dei dati.

`+ void logoutUser();`
Effettua il logout dell'utente.

`+ void sendMessage(String destinatario, String message);`
Invia un messaggio all'utente indicato dalla stringa `destinatario`.

`+ void eraseMessageAll();`
Elimina tutti i messaggi della casella di posta dell'utente.

4.1.1.2 IDataUserLogic

Funzione:
Interfaccia che rappresenta l'accesso agli oggetti che gestiscono il recupero e la modifica dei dati personali dell'utente, verificandone la conformità e inoltrando le richieste di modifica alla componente dedita alla comunicazione con il server$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.logicUser.DataUserLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.serverComUser.WebSocketUser`;

`mytalk.client.view.user.LogUser`.

Metodi:
`+ void logoutUser();`
Effettua il logout dell'utente.

`- void getDataUser();`
Richiama il metodo `getUserData` di `WebSocketUser` che recupera tutti i dati dell'utente.

`- void checkNewData(Vector<String> userInput);`
Controlla i dati inseriti dall'utente.

`+ void loadViewCommunication();`
Richiama la $GUI_{|g|}$ relativa alla comunicazione.

4.1.1.3 ILogUserLogic

Funzione:
Interfaccia che rappresenta l'accesso alle classi logiche che hanno il compito di gestire il login e logout dell'utente al servizio.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.logicUser.LogUserLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.LogUser`.

Metodi:
`+ void validateData(Vector<String> userInput);`
Memorizza l'input dell'utente nel campo dati `loginInput` e richiama il metodo `checkFields()` per controllare i dati inseriti dall'utente. Invoca il metodo `resetErrorMessage()` per resettare il messaggio di errore se ripetuto più volte.

`+ void loadViewRegister();`
Richiama la $GUI_{|g|}$ relativa alla registrazione.

4.1.1.4 IRegisterLogic

Funzione:
Interfaccia che rappresenta l'accesso alle classi logiche che hanno il compito di gestire i dati di registrazione inseriti dall'utente nella View.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.logicUser.RegisterLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.Register`.

Metodi:
`+ void validateData(Vector<String> userInput);`
Memorizza l'input dell'utente nel campo dati `loginInput` e richiama il metodo `checkFields()` per controllare i dati inseriti dall'utente. Invoca il metodo `resetErrorMessage()` per resettare il messaggio di errore in caso di errori ripetuti.

`+ void loadViewLogUser();`
Richiama la $GUI_{|g|}$ relativa all'autenticazione dell'utente.

4.1.1.5 IUpdateViewLogic

Funzione:
Interfaccia che offre i metodi da inoltrare alla view in seguito agli aggiornamenti effettuati dal presenter.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`;

`mytalk.client.presenter.user.logicUser.DataUserLogic`;

`mytalk.client.presenter.user.logicUser.LogUserLogic`;

`mytalk.client.presenter.user.logicUser.RegisterLogic`;

`mytalk.client.presenter.user.serverComUser.WebSocketUser`.

Metodi:
`+void loginResult(boolean loginSucces, Vector<String> loginInput);`
Crea il $cookie_{|g|}$ di sessione contenente il nome dell'utente, altrimenti imposta un opportuno messaggio di errore. Successivamente, aggiorna la View di conseguenza.

`+ void updateViewLogUser(boolean update, String message);`
Richiama il metodo `updateViewLogUser` della classe `PageUserView`.

`+void notifyCall(String utente);`
Richiama il metodo `notifyExternalCall` della classe `PageUserView` al fine di informare l'utente che vi è una chiamata in entrata.

`+ void notifyRefuseCall(String utente);`
Richiama il metodo `notifyRefuseCall` della classe `PageUserView` al fine di

informare l'utente che la sua chiamata è stata rifiutata.

`+ void setUserList(Vector<String> listaUtenti);`
Richiama il metodo `setUsersList` della classe `PageUserView` al fine di impostare la lista degli utenti registrati al servizio.

`+ void checkLoggedUser();`
Controlla che l'utente sia autenticato.

`+ void loadViewRegister();`
Richiama il metodo `loadViewRegister` della classe `PageUserView` al fine di caricare la schermata per la registrazione di un nuovo utente.

`+ void loadViewLogUser();`
Richiama il metodo `loadViewLogUser` della classe `PageUserView` al fine di caricare la schermata per il login dell'utente.

`+ void updateViewRegister(boolean update, Vector<String> messages);`
Richiama il metodo `updateViewRegister` della classe `PageUserView` al fine di informare l'utente sull'esito negativo e sui relativi errori della registrazione al servizio precedentemente inviata.

`+ void registerResult(boolean registerSuccess);`
Riceve l'esito della registrazione.

`+ void removeCookies();`
Rimuovere il cookie$_{|g|}$ relativo alla sessione corrente.

`+ void loadViewUserData();`
Richiama il metodo `loadViewUserData` della classe `PageUserView` al fine di caricare la schermata che permette all'utente di gestire i propri dati personali.

`+ void setLocalVideo(String url);`
Richiama il metodo `setLocalVideo` della classe PageUserView per visualizzare il video locale all'interno della pagina HTML$_{|g|}$.

`+ void setUsernameLabel();`
Richiama il metodo `setUsernameLabel` della classe `PageUserView`. Passandogli il nome utente recuperato dal cookie$_{|g|}$ di sessione.

`+ void loadViewCommunication();`
Richiama il metodo `loadViewCommunication` della classe `PageUserView` per caricare la schermata che permette all'utente di inviare e ricevere comunicazioni
.

`+ void resultDataUser(boolean update);`
Riceve l'esito della modifica dei dati e richiama il metodo `updateViewDataUser` passandogli l'esito e il vector contenente i messaggi d'errore.

`+ void updateViewDataUser(boolean update, Vector<String> messages);`
Il metodo ha lo scopo di aggiornare la schermata dell'utente con il risultato della precedente richiesta di modifica dei dati. Per far ciò richiama il metodo `updateViewDataUser` della classe `PageUserView`, dopo aver richiamato il metodo `setUsernameLabel`.

`+ void setUserDataLabel(Vector<String> userData);`
Richiama il metodo `setUserDataLabel` della classe `PageUserView` per impostare nell'interfaccia grafica il nome dell'utente attualmente connesso.

`+ void callActive(String remoteVideo, String nome);`
Il metodo ha lo scopo di impostare l'url dello stream condiviso dall'utente remoto, in modo da visualizzarlo all'interno della pagina HTML$_{|g|}$, il metodo inoltre imposta la label che mostra all'utente lo username dell'utente con cui si sta comunicando. Il metodo richiama il metodo `callActive` della classe `PageUserView`.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Richiama l'aggiornamento della GUI$_{|g|}$ del pannello di visualizzazione delle statistiche con i dai passati come parametro.

`+ void updatePanelSearch(boolean b, String user);`
Aggiorna il pannello di ricerca dell'utente tramite nome utente o indirizzo IP$_{|g|}$.

`+ void setSegreteria(Vector<String> messaggi);`
Richiama il metodo dell'oggetto `pageUserView void setSegreteria(Vector<String> messaggi)` al fine di permettere la visualizzazione nell'interfaccia grafica dei messaggi in segreteria.

4.1.2 Package mytalk.client.presenter.user.logicUser

4.1.2.1 CommunicationLogic

Funzione:
La classe permette di effettuare e ricevere chiamate e gestione degli stream$_{|g|}$ audio e video. Comunica con il Model e aggiorna opportunamente la View visualizzando messaggi per l'utente ed aggiornando gli stream$_{|g|}$.

Relazioni con altre componenti:
implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iLogicUser.ICommunicationLogic`.

Usa le classi:

`mytalk.client.presenter.user.serverComUser.WebSocketUser`;

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`.

tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic`;

`mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`.

Attributi:
- `IUpdateViewLogic updateView`: riferimento alla classe `UpdateViewLogic` che permette di inviare richieste di aggiornamento dell'interfaccia grafica.

- `IWebSocketUser webSocket`: riferimento alla classe `WebSocketUser` che permette di inviare comunicazioni al server$_{|g|}$ tramite l'interfaccia `IWebSocketUser`.

- `IPeerConnection connessione`: rifermento alla classe `PeerConnection` che permette di instaurare, gestire e chiudere una comunicazione.

- `IMediaStream streamLocale`: riferimento alla classe `MediaStream` che permette di richiedere l'accesso alla videocamera dell'utente e di convertire l'oggetto ottenuto in un'URL$_{|g|}$ visualizzabile in una pagina web$_{|g|}$.

- `String Ricevente`: riferimento ad un oggetto stringa che ha il compito di memorizzare il nome utente di un utente con cui si sta comunicando.

- `Timer stats`: riferimento ad un oggetto di tipo `Timer` che serve a campionare le statistiche in maniera ripetuta e di inviarle all'interfaccia grafica.

Metodi:
`+ CommunicationLogic(IPageUserView v, IWebSocketUser w);`
Inizializza il campo dati connessione degli oggetti `updateView` e `webSocket`, inizializza gli oggetti `connessione` e `streamLocale` tramite i rispettivi costruttori di default e inizializza l'oggetto `statis` di tipo `Timer`.

`+ void call(String utente);`
Viene richiamato quando un utente ne vuole richiamare un altro. Per effettuare la chiamata richiama il metodo dell'oggetto `WebSocketUser call(String utente, String offer)`, passandogli come parametri il nome utente e la descrizione della sessione locale (che è di tipo `offer`). La descrizione della sessione locale viene creata invocando il metodo della classe `PeerConnection inizialize()` e viene reperita già serializzata in formato JSON invocando il metodo della classe `PeerConnection getOffer()`.

La descrizione della sessione locale è un oggetto delle `API WebRTC` che contiene informazione riguardanti i media condivisi e il protocollo utilizzato per stabilire la connessione peer-to-peer (nel caso delle WebRTC$_{|g|}$ è il protocollo ICE).

`+ void accept(String utente);`
Viene richiamato quando si vuole accettare una chiamata da parte dell'utente identificato dalla stringa `utente` passata come parametro attuale. Vi è quindi stata in precedenza una chiamata ricevuta da parte dell'utente passato come parametro e questa chiamata è stata accettata.
Il metodo, per segnalare al server$_{|g|}$ e all'utente l'accettazione della chiamata, richiama il metodo `accept(String utente, String answer)` dell'oggetto `Socket` di tipo `WebSocketUser`, passandogli come parametri, oltre al nome dell'utente che ha chiamato, anche la descrizione locale (che sarà di tipo `answer`) estratta tramite l'invocazione del metodo `getAnswer()` della classe `PeerConnection`.
La descrizione della sessione locale è un oggetto delle `API WebRTC` che contiene informazione riguardanti i media condivisi e il protocollo utilizzato per stabilire la connessione peer-to-peer (nel caso delle WebRTC$_{|g|}$ è il protocollo ICE).

`+ String getLocalURL();`
Restituisce l'URL$_{|g|}$ associato allo stream$_{|g|}$ locale. Per ottenere tale URL$_{|g|}$ il metodo richiama il metodo `localURL` dell'oggetto `StreamLocale`.

`+ void close(int value);`
Chiude eventuali comunicazioni in atto. Per chiudere la comunicazione richiama il metodo dell'oggetto `connessione close()`. Prima di richiamarlo invia le statistiche al server$_{|g|}$ attraverso il metodo dell'oggetto `Socket sendStats(String ricevente, vector<Double> statistiche)`.

`+ void refuse(String utente);`
Viene richiamato quando si vuole rifiutare una chiamata da parte dell'utente identificato dalla stringa `utente` passata come parametro attuale. Per segnalare al server$_{|g|}$ e all'utente il rifiuto della chiamata viene richiamato il metodo `refuse(String utente)` dell'oggetto `Socket` di tipo `WebSocketUser`.

`+ void getStats();`
Quando viene richiamato invia alla View le statistiche estrapolate dall'oggetto `Connessione` richiamando il metodo `getStats()`. Per inviare le statistiche alla GUI$_{|g|}$ viene richiamato il metodo `updateFormInfoChiamata(Vector<Double> statistiche)` dell'oggetto `updateViewLogic`
.

`+ void acceptedCall(String answer, String utente);`
Notifica alla classe che la chiamata dall'altro lato della comunicazione è stato accettata e che si può procedere con l'impostazione della sessione remota, passata come parametro in formato JSON$_{|g|}$.
Il metodo assegna al campo dati `Ricevente` il valore della stringa passata come

parametro al metodo, imposta la descrizione remota richiamando il metodo `setRemoteAnswer(String answer)`, recupera gli `IceCandidate` dall'oggetto `connessione` tramite il metodo `getCandidates()`. Invia l'utente identificato dal parametro formale `utente` tramite `WebSocket` al server$_{|g|}$ richiamando il metodo `sendOffer(String utente, String candidati)`.

`- void sendOffer(String utente, String candidati);`
Invia gli `IceCandidate` di comunicazione costituiti da oggetti `RTCICeCandidate` all'oggetto `Socket` che poi provvederà ad inviarlo al server$_{|g|}$.
Il metodo richiama il metodo dell'oggetto `Socket sendOffer(utente, candidati)` che poi provvederà ad inviare gli `IceCandidate` codificati in formato JSON$_{|g|}$al server$_{|g|}$, il quale girerà poi il messaggio all'utente ricevente.

`- void sendAnswer(String mittente, String candidate);`
Invia gli `IceCandidate` di comunicazione costituiti da oggetti `RTCIceCandidate` all'oggetto `Socket` che poi provvederà ad inviarlo al server$_{|g|}$.
Viene richiamato il metodo dell'oggetto `Socket sendAnswer(utente, candidati)` che provvederà ad inviare gli `IceCandidate` codificati in formato JSON$_{|g|}$ al server$_{|g|}$, il quale girerà poi il messaggio all'utente ricevente.

`+ void receivedSessionPerformed(Vector<String> candidate);`
Imposta la descrizione di sessione remota di tipo `answer` giunta tramite il `WebSocket`, in modo che sia possibile iniziare la comunicazione.
Viene richiamato il metodo dell'oggetto `Connessione` di tipo `PeerConnection receivedCommunication(JavaScriptObject description)` per impostare la descrizione di sessione.

`+ void receivedSession(String mittente, Vector<String> candidate);`
Aggiunge i candidati ICE alla connessione richiamando ripetutamente il metodo della classe `PeerConnection addCandidate(String candidate)`, successivamente reperisce gli `IceCandidate` dalla connessione tramite `PeerConnection getCandidates()` e richiama il metodo `sendAnswer(String utente, String candidates)` per inviare gli oggetti di tipo `RTCIceCandidate` in risposta all'utente che ha inviato i suoi candidati in precedenza.

`+ void callEnter(String offer);`
Il metodo ha il compito di richiamare il metodo della classe `PeerConnection answer(String offer)`, tramite il quale il sistema riesce a creare una descrizione di sessione locale di tipo `answer` in relazione alla descrizione di sessione remota di tipo `offer` passata come riferimento al metodo.

`- void setCallActive(String utente);`
Il metodo prima recupera l'URL remoto dalla classe `PeerConnection` tramite il metodo `getRemoteURL()`, poi richiama il metodo della classe `UpdateViewLogic callActive(String url, String utente`, che imposterà lo stream$_{|g|}$ remoto all'interno dell'interfaccia grafica e imposterà il nome dell'utente con il quale

si sta comunicando. Il richiamo di questo metodo dà inizio alla comunicazione vera e propria.

`+ void logoutUser();`
Effettua il logout dell'utente: chiama i metodi per rimuovere il cookie$_{|g|}$ associato all'utente e aggiornare la View.

`+ void search(String type, String user);`
Invia al server$_{|g|}$ una richiesta riguardante l'esistenza di un utente registrato al servizio. Se la ricerca di reperimento dell'username avviene tramite IP$_{|g|}$, viene richiamato il metodo dell'oggetto `webSocketUser searchUserByIp(String ip)`, se invece la ricerca avviene per inserimento di una e-mail viene richiamato il metodo della classe `webSocketUser searchUserByEMail(String user)`.

`+ void setUserList();`
Richiede al server$_{|g|}$, tramite `WebSocket`, la lista degli utenti registrati al servizio.

`+ void setUsernameLabel();`
Imposta la label del nome dell'utente autenticato richiamando il metodo `mytalk.client.presenter.administrator.logicUser.UpdateViewLogic.setUsernameLabel()`.

`+ void logoutUser();`
Effettua il logout dell'utente: richiama i metodi per rimuovere il cookie$_{|g|}$ associato all'utente e aggiornare la View.

`+ void sendMessage(String destinatario, String message);`
Richiamando il metodo dell'oggetto `webSocketUser sendMessage(String destinatario, String message)` invia un messaggio all'utente indicato dalla stringa `destinatario`.

`+ void eraseMessageAll();`
Richiamando il metodo dell'oggetto `webSocketUser void eraseMessageAll()` elimina tutti i messaggi della casella di posta dell'utente.
`+ void loadViewUserData()`
Richiama la GUI$_{|g|}$ relativa alla gestione dei dati richiamando il metodo `mytalk.client.presenter.administrator.logicUser.UpdateViewLogic.loadViewUserData()`.

4.1.2.2 DataUserLogic

Funzione:
La classe controlla che i dati che l'utente vuole modificare siano corretti tramite l'utilizzo di espressioni regolari e richiama opportuni metodi della classe che gestisce i `WebSocket` per poter modificare in maniera persistente i dati

dell'utente autenticato.

Relazioni con altre componenti:
implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iLogicUser.IDataUserLogic`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`;

`mytalk.client.presenter.user.serverComUser.WebSocketUser`;

`mytalk.client.model.localDataUser.ManageCookies`.

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic`;

`mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`.

Attributi:
- `Vector<String> userInput`: contiene i nuovi dati inseriti dall'utente.

- `boolean dataOk`: contiene l'esito dei controlli preliminari sui nuovi dati inseriti.

- `Vector<String> errorMessage`: contiene eventuali messaggi di errori che si sono verificati nei controlli preliminari dei dati inseriti.
- `WebSocketUser webSocket`: è un riferimento all'oggetto `WebSocketUser`. Tale riferimento permette di inviare messaggi al server$_{|g|}$.

- `IUpdateViewLogic updateViewLogic`: è un riferimento all'oggetto di tipo `UpdateViewLogic`. Tale riferimento permette di richiamare metodi che poi andranno a modificare la View per visualizzare informazioni per l'utente.

Informazioni generali sui seguenti metodi: tutte le stringhe prima della loro gestione sono state trattate con il metodo `trim()` che rimuove gli spazi bianchi all'inizio ed alla fine della stringa.

Metodi:
`+ DataUserLogic(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocket);`
Il costruttore inizializza i campi dati dell'oggetto con i parametri passatigli.

`+ void logoutUser();`
Effettua il logout dell'utente: richiama i metodi per rimuovere il cookie$_{|g|}$ associato all'utente e aggiornare la View.

`+ void getDataUser();`
Richiama il metodo `getUserData()` della classe `WebSocketUser`, il quale recupera tutti i dati dell'utente.

`+ void checkNewData(Vector<String> input)`
Controlla che i dati inseriti dall'utente siano corretti, per eseguire i controlli richiama il metodo `checkFields()`. In caso affermativo richiama il metodo `changeDataUser` della classe `WebSocketUser` per aggiornare i dati dell'utente, altrimenti aggiorna la View degli errori.

`+ void checkFields();`
Imposta il vettore contenente i messaggi di errore relativi ad ogni campo, richiama i metodi per controllare i vari campi ed imposta il campo dati `loginOk` a `true` se non vengono riscontrati errori, altrimenti a `false`. I controlli eseguiti per i vari campi sono:

- l'indirizzo e-mail deve avere un formato appropriato;
- il nome e il cognome devono essere stringhe;
- il numero di telefono deve essere lungo al massimo 10 caratteri tutti numerici.

`- void checkEMail();`
Richiama il metodo di `CommonFunctions` per controllare che l'indirizzo e-mail abbia un formato appropriato.

`- void checkStringData();`
Richiama il metodo di `CommonFunctions` per controllare che il nome e il cognome siano composti da stringhe alfabetiche.

`- void checkPassword();`
Richiama il metodo di `CommonFunctions` per controllare la password e l'uguaglianza della nuova password e della conferma della password.

`- void checkOptional();`
Richiama i metodi di `CommonFunctions` per controllare il nome dell'azienda e il numero telefonico.

`+ void loadViewCommunication();`
Richiama la $GUI_{|g|}$ relativa alla comunicazione richiamando il metodo `mytalk.client.presenter.administrator.logicUser.UpdateViewLogic.loadViewCommunication()`.

4.1.2.3 LogUserLogic

Funzione:
la classe ha il compito di inoltrare la richiesta di login alla classe

`IWebSocketUser`, che comunica con il server$_{|g|}$. Richiama i metodi di `IUpdateViewLogic` per notificare la View dell'esito della verifica delle credenziali ed effettua il logout dell'utente distruggendo i dati relativi alla sessione.

Relazioni con altre componenti:
implementa l'interfaccia :

`mytalk.client.iPresenter.iUser.iLogicUser.ILogUserLogic`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.common.CommonFunctions`;

`mytalk.client.presenter.user.serverComUser.WebSocketUser`;

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`;

Tramite le interfacce:

`mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`;

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic`.

Attributi:
- `Vector<String> loginInput`: contiene i dati inseriti dall'utente provenienti dalla View;

- `boolean loginOk`: indica se i dati inseriti dall'utente risultano corretti o meno;

- `String errorMessage` messaggio di errore da ritornare all'utente in caso di problemi nel login;

- `IWebSocketUser webSocketUser`: riferimento alla classe `WebSocketUser`. Permette di richiamare i metodi per l'autenticazione e la terminazione della sessione utente;

- `IUpdateViewLogic updateViewLogic`: riferimento alla classe `UpdateViewLogic`. Permette di richiamare i metodi di aggiornamento della View.

Metodi:
`+ LogUserLogic(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocketUser);`
Costruttore. Inizializza `updateViewLogic` e `webSocketUser` con i valori ricevuti come parametri, crea un nuovo `Vector<String>` vuoto per `loginInput`, inizializza a `true loginOk` e inizializza `errorMessage`.

`+ void validateData(Vector<String> userInput);`
Memorizza l'input dell'utente in `loginInput`, richiama il metodo di `CommonFunctions CorrectInput` e richiama `checkFields` per verificare i dati ricevuti. Se questi sono corretti, richiama `authicateUser` per autenticare l'utente. Infine, utilizza il riferimento `updateViewLogic` per notificare la View dell'esito dell'operazione.

`- void resetErrorMessage();`
Reimposta il messaggio di errore al valore iniziale. Viene richiamato da `checkFields()` per evitare di mostrare messaggi di errore ripetuti all'utente.

`- void checkFields();`
Dopo aver reimpostato `errorMessage` e `loginOk` ai valori iniziali, utilizza i metodi della classe astratta `CommonFunctions` per controllare la correttezza sintattica dell'indirizzo e-mail e della password inseriti dall'utente. In caso di errore, aggiorna il messaggio di errore e imposta `loginOk` a `false`.

`- void authicateUser();`
Autenticazione dell'utente. Richiama il metodo di `webSocketUser` per autenticare l'utente.

`+ void loadViewRegister();`
Richiama la $GUI_{|g|}$ relativa alla registrazione richiamando il metodo `mytalk.client.presenter.administrator.logicUser.UpdateViewLogic.loadViewRegister()`.

4.1.2.4 RegisterLogic

Funzione:
La classe ha il compito di inoltrare la richiesta di registrazione alla classe `IWebSocketUser`, che comunica con il $server_{|g|}$. Richiama inoltre i metodi di `IUpdateViewLogic` per notificare la View dell'esito della verifica delle credenziali. La classe permette all'utente di effettuare la registrazione per usufruire del servizio.

Relazioni con altri metodi:
implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iLogicUser.IRegisterLogic`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.common.CommonFunctions`;

`mytalk.client.presenter.user.serverComUser.WebSocketUser`;

tramite l'interfaccia:

`mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`.

Attributi:
- `Vector<String> userInput`: contiene i dati inseriti dall'utente provenienti dalla View;

- `boolean loginOk`: indica se i dati inseriti dall'utente risultano corretti o meno;

- `String errorMessage`: messaggio di errore da ritornare all'utente in caso di problemi nel login;

- `IWebSocketUser webSocketUser`: riferimento alla classe `WebSocketUser`. Permette di richiamare i metodi per l'autenticazione e la terminazione della sessione utente;

- `IUpdateViewLogic updateViewLogic`: riferimento alla classe `UpdateViewLogic`. Permette di richiamare i metodi di aggiornamento della View.

Metodi:
`+ RegisterLogic(IUpdateViewLogic updateViewLogic, IWebSocketUser webSocketUser);`
Costruttore. Inizializza `updateViewLogic` e `webSocketUser` con i valori ricevuti come parametri, crea un nuovo `Vector<String>` per `registerInput` e lo inizializza con la stringa vuota, inizializza `loginOk` a `true`.

`+ void validateData(Vector<String> userInput);`
Memorizza l'input dell'utente in `registerInput`, richiama il metodo di `CommonFunctions CorrectInput` e richiama `checkFields` per verificare i dati ricevuti. Se questi sono corretti, richiama `webSocketUser.inserisciUtente(userInput)` per registrare l'utente. Infine, utilizza il riferimento `updateViewLogic` per notificare la View dell'esito dell'operazione.

`- void checkFields();`
Richiama i vari metodi corrispondenti alla registrazione per controllare l'input dell'utente.

`- void checkEMail();`
Richiama il metodo di `CommonFunctions` per controllare l'e-mail.

`- void checkStringData();`
Richiama il metodo di `CommonFunctions` per controllare il nome e il cognome.

`- void checkPassword();`
Richiama il metodo di `CommonFunctions` per controllare la password e l'uguaglianza della nuova password e della conferma della password.

`- void checkOptional();`
Richiama i metodi di `CommonFunctions` per controllare il nome dell'azienda e il numero telefonico.

`+ void loadViewLogUser();`
Richiama la $GUI_{|g|}$ relativa all'autenticazione dell'utente richiamando il metodo `mytalk.client.presenter.administrator.logicUser.UpdateViewLogic.loadViewLogUser()`.

4.1.2.5 UpdateViewLogic

Funzione:
Inoltra alla View tutte le richieste di aggiornamento provenienti dal Presenter. Controlla inoltre se l'utente è autenticato e gestisce l'esito delle operazioni di login, registrazione e modifica dati.

Relazioni con altre componenti:
implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iLogicUser.IUpdateViewLogic`.

Usa le classi:

`mytalk.client.presenter.user.logicUser.common.CommonFunctions`;

`mytalk.client.view.user.PageUserView`;

tramite l'interfaccia:

`mytalk.client.iView.iUser.IPageUserView`.

Attributi:
`- IPageUserView pageUserView`: riferimento alla classe `mytalk.client.view.user.PageUserView`. Permette di richiamare i metodi di aggiornamento della View.

Metodi:
`+ UpdateViewLogic(IPageUserView pageUserView);`
Costruttore: inizializza il riferimento `pageUserView` con quello passatogli come parametro.

`+void loginResult(boolean loginSucces, Vector<String> loginInput);`
`SeloginSuccess` vale `true` richiama il metodo di `ManageCookies` per creare

il cookie$_{|g|}$ di sessione contenente il nome dell'utente, altrimenti imposta un opportuno messaggio di errore. Successivamente, aggiorna la View di conseguenza.

`+ void updateViewLogUser(boolean update, String message);`
Richiama il metodo `updateViewLogUser` della classe `PageUserView`.

`+void notifyCall(String s);`
Richiama il metodo `notifyExternalCall` della classe `PageUserView`.

`+ void notifyRefuseCall(String s);`
Richiama il metodo `notifyRefuseCall` della classe `PageUserView`.

`+ void setUserList(Vector<String> listaUtenti);`
Richiama il metodo `setUsersList` della classe `PageUserView`.

`+ void checkLoggedUser();`
Controlla che l'utente sia autenticato. Per farlo prova a recuperare il cookie$_{|g|}$ di sessione richiamando l'apposito metodo della classe `ManageCookies`: se il cookie$_{|g|}$ esiste, l'utente è autenticato.

`+ void loadViewRegister();`
Richiama il metodo `loadViewRegister` della classe `PageUserView`.

`+ void loadViewLogUser();`
Richiama il metodo `loadViewLogUser` della classe `PageUserView`.

`+ void updateViewRegister(boolean update, Vector<String> messages);`
Richiama il metodo `updateViewRegister` della classe `PageUserView`.

`+ void registerResult(boolean registerSuccess);`
Riceve l'esito della registrazione e richiama il metodo `updateViewRegister` con parametro `true` o `false`. Imposta opportunamente il vettore dei messaggi di errore `messages`.

`+ void removeCookies();`
Richiama il metodo `deleteCookies` della classe `ManageCookies` per rimuovere il cookie$_{|g|}$ relativo alla sessione corrente.

`+ void loadViewUserData();`
Richiama il metodo `loadViewUserData` della classe `PageUserView`.

`+ void setLocalVideo(String url);`
Richiama il metodo `setLocalVideo` della classe PageUserView.

`+ void setUsernameLabel();`
Richiama il metodo `setUsernameLabel` della classe `PageUserView`. Gli passa il nome utente recuperato dal cookie$_{|g|}$ di sessione.

`+ void loadViewCommunication();`
Richiama il metodo `loadViewCommunication` della classe `PageUserView`.

`+ void resultDataUser(boolean update);`
Riceve l'esito della modifica dei dati e richiama il metodo `updateViewDataUser` passandogli l'esito e il `Vector` contenente i messaggi d'errore.

`+ void updateViewDataUser(boolean update, Vector<String> messages);`
Richiama il metodo `updateViewDataUser` della classe `PageUserView`, dopo aver richiamato il metodo `setUsernameLabel`.

`+ void setUserDataLabel(Vector<String> userData);`
Richiama il metodo `setUserDataLabel` della classe `PageUserView`.

`+ void callActive(String remoteVideo, String nome);`
Richiama il metodo `callActive` della classe `PageUserView`.

`+ void updateFormInfoChiamata(Vector<Double> dataCall);`
Richiama il metodo `updateFormoInfoChiamata(dataCall)` attraverso il riferimento `pageUserView`.

`+ void updatePanelSearch(boolean b, String user);`
Richiama il metodo `updatePanelSearch(b, user)` attraverso il riferimento `pageUserView`.

`+ void setSegreteria(Vector<String> messaggi);`
Richiama il metodo dell'oggetto `pageUserView void setSegreteria(Vector<String> messaggi)` al fine di permettere la visualizzazione nell'interfaccia grafica dei messaggi in segreteria.

4.1.3 Package mytalk.client.iPresenter.iUser.iCommunication

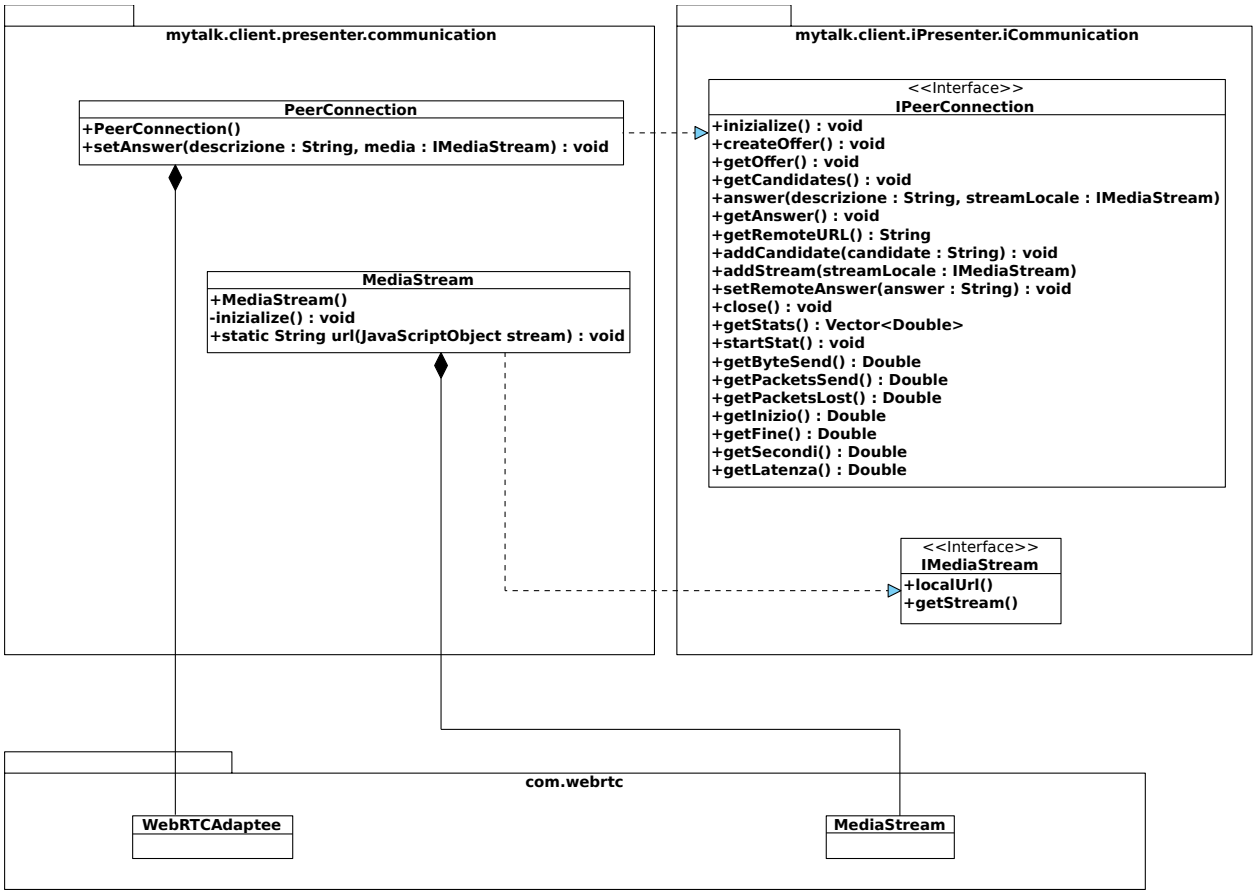

Figura 9: Diagramma delle classi dei package `mytalk.client.iPresenter.iUser.iCommunication` e `mytalk.client.presenter.user.communication`; dettaglio delle classi `IPeerConnction`, `IMediaStream`, `PeerConnction` e `MediaStream`.

4.1.3.1 IMediaStream

Funzione:
Le classi che implementano questa interfaccia hanno il compito di ottenere e gestire l'accesso ai dispositivi hardware$_{|g|}$ che permettono la comunicazione audio e video.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.communication.MediaStream`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Metodi:
`+ String localURL();`
Restituisce l'URL$_{|g|}$ associato ad uno stream$_{|g|}$ locale, altrimenti una stringa inizializzata a `null`.

`+ JavaScriptObject getStream();`
Restituisce l'oggetto JavaScript$_{|g|}$ che rappresenta lo stream$_{|g|}$ locale al quale l'utente ha precedentemente consentito l'acquisizione.

4.1.3.2 IPeerConnection

Funzione:
L'interfaccia fornisce le funzioni fondamentali per la gestione della connessione per gli utenti che vogliono comunicare.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.user.communication.PeerConnection`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Metodi:
`+ void inizialize();`
Inizializza i campi dati della connessione.

`+ void answer(String descrizione, IMediaStream stream);`
Aggiunge uno stream$_{|g|}$ locale alla connessione e crea una risposta che poi sarà inviata ad un utente. Per risposta si intende una descrizione di sessione di tipo `Answer`.

`+ String getAnswer();`
Restituisce un oggetto in un formato assegnabile ad una stringa che descrive la sessione locale da inviare come risposta ad un altro ricevuto in precedenza.

`+ String getCandidates();`
Restituisce una stringa da inviare ad un utente remoto contenente tutti i candidati ICE creati in precedenza.

`+ String getOffer();`
Restituisce un oggetto in un formato assegnabile ad una stringa che descrive la sessione locale da inviare ad un utente remoto.

`+ String getRemoteURL();`
Restituisce l'URL$_{|g|}$ dello stream$_{|g|}$ remoto eventualmente esistente, altrimenti una stringa vuota.

`+ void createOffer();`
Crea una offerta che poi sarà inviata ad un utente. Per offerta si intende una descrizione di sessione di tipo `Offer`.

`+ void close();`
Chiude la comunicazione eventualmente in atto.

`+ Vector<Double> getStats();`
Restituisce un vettore di `Double` che rappresenta le statistiche da visualizzare nella GUI$_{|g|}$ o da inviare al server$_{|g|}$.

`+ void receivedComunication(Vector<String> candidate);`
Imposta i candidati ICE$_{|g|}$ all'interno dell'oggetto che gestisce la connessione.

`+ void addCandidate(String candidate);`
Imposta il candidato ICE$_{|g|}$ all'interno dell'oggetto che gestisce la connessione.

`+ void addStream(IMediaStream stream);`
Aggiunge uno stream$_{|g|}$ locale all'oggetto che gestisce la connessione.

`+ void setRemoteAnswer(String answer);`
Imposta una descrizione remota di tipo `answer` all'oggetto che gestisce la connessione.

`+ void startStats();`
Invia un segnale all'oggetto che gestisce la connessione che da inizio al campionamento delle statistiche.

`+ Double getByteSend();`
Ritorna il numero di byte inviati dagli stream$_{|g|}$ audio e video (la somma dei due).

`+ Double getPacketsSend();`
Ritorna il numero dei pacchetti inviati dagli stream$_{|g|}$ audio e video.

`+ Double getPacketsLost();`
Ritorna il numero di pacchetti persi nel corso della comunicazione dagli stream$_{|g|}$ audio e video.

`+ Double getInizio();`
Ritorna il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970

all'inizio della comunicazione.

`+ Double getFine();`
Ritorna il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970 alla fine della comunicazione.

`+ Double getSecondi();`
Ritorna il numero di secondi intercorsi dall'inizio della comunicazione alla chiamata del metodo.

`+ Double getLatenza();`
Ritorna la latenza$_{|g|}$ dello stream$_{|g|}$ audio al momento della chiamata del metodo.

4.1.4 Package mytalk.client.presenter.user.communication

4.1.4.1 MediaStream

Funzione:
La classe gestisce i media locali e richiede il permesso di accesso all'hardware$_{|g|}$ all'utente.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iCommunication.IMediaStream`.

Usa le classi:

`APIgetUserMedia`;

tramite l'interfaccia:

`mytalk.client.iPresenter.iUser.iCommunication.IMediaStream`.

Attributi:
- `JavaScriptObject localStream`: l'oggetto garantisce l'accesso all'hardware$_{|g|}$ locale, se l'oggetto è inizializzato allora l'utente ha acconsentito all'accesso all'hardware$_{|g|}$ locale(videocamera e microfono oppure solo microfono).

- `String localURL`: la stringa contiene l'URL$_{|g|}$ associato allo stream$_{|g|}$ locale al quale l'utente ha garantito l'accesso, se la stringa è uguale a null allora o l'utente ha rifiutato la richiesta dell'applicazione di accedere all'hardware$_{|g|}$ oppure non ha ancora dato il consenso.

Metodi:
`+ void inizialize();`
Inizializza lo stream$_{|g|}$ con un oggetto che rappresenta e consente l'accesso all'hardware$_{|g|}$ locale. Per far ciò richiama il metodo dell'oggetto `JavaScript window.GetUserMedia` che richiede all'utente l'accesso all'hardware$_{|g|}$. Se l'utente acconsente alla richiesta d'accesso allora richiama un metodo che memorizza l'oggetto che rappresenta lo stream$_{|g|}$ e lo converte in un URL$_{|g|}$.

`+ String localURL();`
Restituisce l'URL$_{|g|}$ associato ad uno stream$_{|g|}$ locale, altrimenti una stringa inizializzata a `null`.

`+ JavaScriptObject getStream();`
Restituisce l'oggetto JavaScript$_{|g|}$ che rappresenta lo stream$_{|g|}$ locale al quale l'utente ha precedentemente consentito l'acquisizione.

4.1.4.2 PeerConnection

Funzione:
La classe permette di gestire i dati delle `API WebRTC`$_{|g|}$ e di stabilire connessioni VoIP$_{|g|}$ peer-to-peer$_{|g|}$.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iPresenter.iUser.iCommunication.IPeerConnection`.

Usa le classi:

`WebRTCAdaptee`.

Attributi: - `JavaScriptObject WebRTC`: permette di aprire, gestire e chiudere una comunicazione peer-to-peer$_{|g|}$.

Metodi:
`+ void inizialize();`
Richiama il metodo `inizialize()` della classe `WebRTCAdaptee`. In questo modo l'oggetto che gestisce la connessione viene inizializzato e ne vengono registrati gli eventi utilizzati dall'applicazione.

`+ void addStream(IMediaStream stream);`
Aggiunge uno stream$_{|g|}$ locale alla connessione, richiama il metodo `addstream` della classe `WebRTCAdaptee` per effettuare l'aggiunta dello stream$_{|g|}$, in questo modo lo stream può essere condiviso con l'utente con cui si comunica.

`+ String getRemoteURL();`
Restituisce l'URL$_{|g|}$ dello stream$_{|g|}$ remoto richiamando il metodo della classe

`WebRTCAdaptee getRemoteUrl()`, in questo modo lo stream è visualizzabile all'interno di una pagina html5.

`+ void createOffer();`
Richiama il metodo della classe `WebRTCAdaptee offer()`. In questo modo viene creata una descrizione della sessione locale da inviare ad un altro utente di tipo `offer`, la descrizione di sessione contiene informazioni come gli stream condivisi e il protocollo utilizzato per creare il canale di comunicazione VoIP.

`+ void answer(String remoteOffer, MediaStream streamLocale);`
Richiama il metodo della classe `WebRTCAdaptee answer(String remoteOffer, MediaStream streamLocale)` passandogli come parametri la descrizione remota di tipo `offer` e l'oggetto che rappresenta lo stream$_{|g|}$ locale. Successivamente vengono impostati lo stream$_{|g|}$ locale, la descrizione remota e la descrizione locale appena creata, la descrizione di sessione contiene informazioni come gli stream$_{|g|}$ condivisi e il protocollo utilizzato per creare il canale di comunicazione VoIP.

`+ void close();`
Chiude la comunicazione eventualmente in atto, richiamando il metodo della classe `WebRTCAdaptee close()`, reinizializzando i campi dati a null.

`+ String getAnswer();`
Restituisce un oggetto in un formato JSON$_{|g|}$ che descrive la sessione locale da inviare come risposta ad un altro ricevuto in precedenza.

`+ String getOffer();`
Restituisce un oggetto in un formato JSON$_{|g|}$ che descrive la sessione locale da inviare ad un utente remoto.

`+ String getCandidates();`
Restituisce una stringa da inviare ad un utente remoto contenente tutti i candidati ICE creati in precedenza.

`+ void receivedComunication(Vector<String> candidate);`
Imposta i candidati ICE (oggetti WebRTC che contengono informazioni su un peer, informazioni come indirizzo IP, porta del sistema utilizzabile per la comunicazione e protocollo di comunicazione) all'interno dell'oggetto che gestisce la connessione. Per far ciò richiama per ogni elemento dell'array passato come parametro il metodo `addCandidate(String ice)` dell'oggetto della classe `WebRTCAdaptee`.

`- void setAnswer(String descrizione,IMediaStream media);`
Imposta un oggetto `RTCSessionDescrption` di tipo `answer` all'interno dell'oggetto di tipo `webkitRTCPeerConnection`, interno alla classe `WebRTCAdaptee`, in questo modo viene impostata la descrizione di sessione inviata dall'altro

utente, mancano tuttavia gli ICE candidate per iniziare la connessione, tali candidati verranno inviati successivamente.

`+ void setRemoteAnswer(String answer);`
Imposta una descrizione remota di tipo 'answer' all'oggetto che gestisce la connessione.

`+ Vector<Double> getStats();`
Restituisce un vettore di `Double` che rappresentano le statistiche da visualizzare nella $GUI_{|g|}$ o da inviare al $server_{|g|}$. Per estrapolare le statistiche si serve del metodo della classe `RTCPeerConnection getStats()` dal quale riceve un array associativo che contiene le statistiche.

`+ void addCandidate(String candidate);` Il metodo riceve in input un oggetto di tipo `RTCIceCandidate` codificato in formato $JSON_{|g|}$ e invoca il metodo della classe `WebRTCAdaptee addCandidate()` con lo scopo di aggiungere il candidato ICE all'oggetto che rappresenta la connessione.

`+ void startStats();`
Invia un segnale all'oggetto che gestisce la connessione che dà inizio al campionamento delle statistiche. Il segnale è rappresentato dalla chiamata del metodo `startStat()` dell'oggetto `WebRTCAdaptee`.

`+ Double getByteSend();`
Invoca `WebRTCAdaptee getByteSend()` dal quale riceve il numero di byte inviati dagli $stream_{|g|}$ audio e video(la somma dei due).

`+ Double getPacketsSend();`
Invoca `WebRTCAdaptee getPacketsSend()` dal quale riceve il numero dei pacchetti inviati dagli $stream_{|g|}$ audio e video.

`+ Double getPacketsLost();`
Invoca `WebRTCAdaptee getPacketsSend()` dal quale riceve il numero di pacchetti persi nel corso della comunicazione dagli $stream_{|g|}$ audio e video.

`+ Double getInizio();`
Invoca `WebRTCAdaptee getDataInizio()` dal quale riceve il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970 all'inizio della comunicazione.

`+ Double getFine();`
Invoca `WebRTCAdaptee getDataFine()` dal quale riceve il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970 alla fine della comunicazione.

`+ Double getSecondi();`
Invoca `WebRTCAdaptee getSeconds()` dal quale riceve il numero di secondi

intercorsi dall'inizio della comunicazione alla chiamata del metodo.

`+ Double getLatenza();`
Invoca `WebRTCAdaptee` `getJitter()` dal quale riceve la latenza$_{|g|}$ dello stream$_{|g|}$ audio al momento della chiamata del metodo.

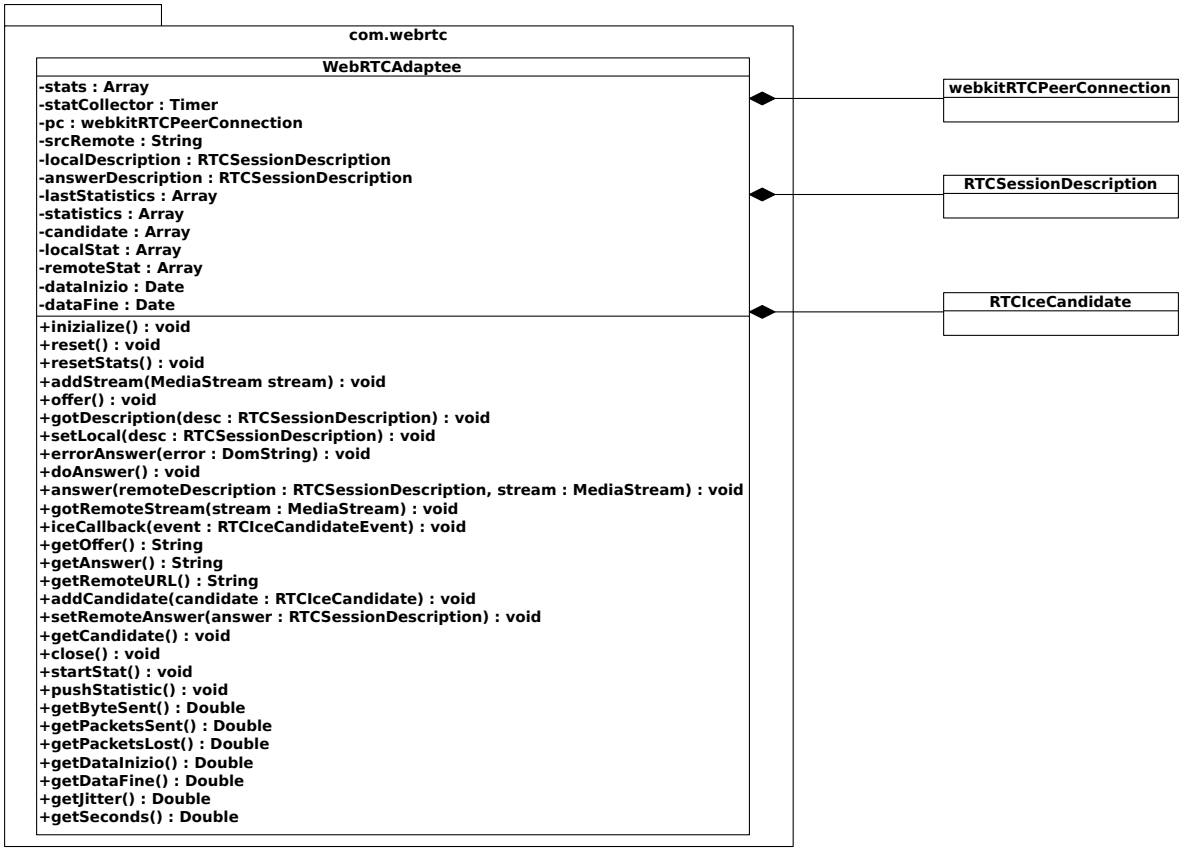

Figura 10: Diagramma delle classi del package `com.webrtc`; dettaglio della classe `WebRTCAdaptee`.

4.1.4.3 WebRTCAdaptee

Funzione:
La classe permette di gestire i dati delle `API WebRTC`$_{|g|}$ e di stabilire connessioni VoIP$_{|g|}$ peer-to-peer$_{|g|}$.

Relazioni con altre componenti:
Usa le classi:

`API WebRTC`;

`JavaScript.Date`;

`JavaScript.Array`.

Attributi:
- `Array stats`: riferimento all'oggetto che raccoglie le statistiche.

- `Array statsCollector`: riferimento all'oggetto che contiene lo storico di tutte le statistiche campionate dalla creazione della connessione.

- `webkitRTCPeerConnection pc`: riferimento, possibilmente nullo, all'oggetto `RTCPeerConnection` che gestisce la connessione.

- `String srcRemote`: $URL_{|g|}$ dello $stream_{|g|}$ remoto, in modo da permetterne la visualizzazione tramite l'interfaccia grafica. Il riferimento è nullo se l'$URL_{|g|}$ remoto non è ancora stato estratto dalla descrizione remota.

- `String localDescription`: il campo contiene, se diverso da `null`, la descrizione della sessione locale codificata secondo il formato $JSON_{|g|}$.

- `String answerDescription`: il campo contiene, se diverso da `null`, la descrizione della sessione locale di tipo `answer` codificata secondo in formato $JSON_{|g|}$.

- `Array lastStatistics`: il campo contiene le ultime statistiche campionate.

- `Array candidate`: l'array contiene, se non vuoto, i candidati ICE registrati all'atto della creazione della sezione locale.

- `int DataInizio`: millisecondi intercorsi dalla mezzanotte del 01/01/1970 all'inizio della chiamata.

- `int DataFine`: millisecondi intercorsi dalla mezzanotte del 01/01/1970 alla fine della chiamata.

Metodi:
`+ void inizialization();`
Inizializza il campo dati `pc` e ne registra gli eventi.

`+ void addStream(Stream stream);`
Aggiunge uno $stream_{|g|}$ locale alla connessione.

`+ void offer();`
Crea un oggetto `RTCSessionDescription` di tipo `offer`. Se la creazione ha successo viene chiamato il metodo di callback `gotDescription(RTCSessionDescription description)`.

`+ void gotDescription(RTCSessionDescription description);`
Salva il parametro `description` come una stringa JSON$_{|g|}$ nel campo dati `localDescription` e imposta l'oggetto come descrizione della sessione locale tramite la chiamata al metodo `SetLocalDescription` della classe `webkitRTCPeerConnection`.

`+ String getRemoteURL();`
Restituisce l'URL$_{|g|}$ dello stream$_{|g|}$ remoto se esiste, `null` altrimenti.

`+ void iceCallback(event e);`
Metodo di callback che viene invocato quando viene creato un oggetto `RTCIceCandidate` in seguito alla creazione di un oggetto `RTCSessionDescription`. Il metodo converte l'oggetto `RTCIceCandidate` in una stringa JSON$_{|g|}$ e lo salva nell'array `candidate`.

`+ void answer(RTCSessioneDescription descrizione, Stream stream);`
Reinizializza il campo dati `pc`, aggiunge all'oggetto `pc` lo stream$_{|g|}$ locale passato come parametro di nome `stream`. Imposta come descrizione di sessione remota l'oggetto `descrizione` passato come parametro e crea una descrizione di sessione tramite la chiamata al metodo `doAnswer()`.

`+ void doAnswer();`
Richiama il metodo `createAnswer()` delle WebRTC$_{|g|}$ sull'oggetto `pc` al fine di creare una descrizione di sessione di tipo `answer`. Se la creazione dell'oggetto ha successo viene richiamato il metodo di callback `setLocal(RTCSessionDescription description)`, altrimenti viene richiamato il metodo `errorAnswer()`.

`+ void setLocal(RTCSessionDescription descrizione);`
Metodo di callback che viene invocato se la creazione di un oggetto `RTCSessionDescription` di tipo `answer` ha avuto successo. Il metodo salva l'oggetto in formato JSON$_{|g|}$ nel campo dati `answerDescription` e imposta l'oggetto come descrizione della sessione locale tramite la chiamata al metodo `SetLocalDescription` della classe `webkitRTCPeerConnection`.

`+ void errorAnswer(string answer);`
Il metodo viene invocato se vi è stato un errore nella creazione dell'oggetto `RTCSessionDescription` di tipo `answer`.

`+ void close();`
Chiude la comunicazione eventualmente in atto richiamando il metodo della classe `RTCPeerConnection close()` sull'oggetto `pc`. Ferma il campionamento delle statistiche e dopo 2 secondi dalla chiusura reinizializza tutti i campi dati per prepararli ad una nuova eventuale comunicazione.

`+ void addCandidate(String candidate);` Aggiunge un candidato ICE alla connessione chiamando il metodo della classe `webkitRTCPeerConnection`

`addIceCandidate(RTCIceCandidate iceCandidate)`.

`+ void startStats();`
Imposta un intervallo che campiona le statistiche una volta ogni secondo. Per ottenere le statistiche si serve della funzione delle WebRTC$_{|g|}$ `getStats()` che viene invocata sull'oggetto `pc` di tipo `webkitRTCPeerConnection`.

`+ void setRemoteAnswer(String answer);`
Il metodo riceve in input un oggetto `RTCSessioneDescription` di tipo `answer` e lo imposta come descrizione remota tramite il metodo dell'oggetto `webkitRTCPeerConnection setRemoteDescription(RTCSessionDescription description)`.

`+ Double getByteSend();`
Ritorna il numero di byte inviati dagli stream$_{|g|}$ audio e video(la somma dei due).

`+ Double getPacketsSend();`
Ritorna il numero dei pacchetti inviati dagli stream$_{|g|}$ audio e video.

`+ Double getPacketsLost();`
Ritorna il numero di pacchetti persi nel corso della comunicazione dagli stream$_{|g|}$ audio e video.

`+ Double getDataInizio();`
Ritorna il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970 all'inizio della comunicazione.

`+ Double geDatatFine();`
Ritorna il numero di millisecondi intercorsi dalla mezzanotte del 01/01/1970 alla fine della comunicazione.

`+ Double getSeconds();`
Ritorna il numero di secondi intercorsi dall'inizio della comunicazione alla chiamata del metodo.

`+ Double getJitter();`
Ritorna la latenza$_{|g|}$ dello stream$_{|g|}$ audio al momento della chiamata del metodo.

`+ void reset();`
Assegna il valore `null` a tutti i campi dati della classe per iniziare una nuova comunicazione.

`+ void resetStats();`
Cancella tutti i campionamenti delle statistiche avvenuti nell'ultima chiamata.

`+ void getOffer();`
Restituisce un oggetto di classe `RTCSessionDescription` di tipo `Offer` in formato $JSON_{|g|}$ se è stato creato in precedenza `null` altrimenti.

`+ void getAnswer();`
Restituisce un oggetto di classe `RTCSessionDescription` di tipo `Answer` in formato $JSON_{|g|}$ se è stato creato in precedenza `null` altrimenti.

`+ void getCandidate();`
Restituisce gli oggetti di classe `RTCICeCandidate` in formato $JSON_{|g|}$ creati in precedenza.

4.1.5 Package mytalk.client.iPresenter.iAdministrator.iLogicAdmin

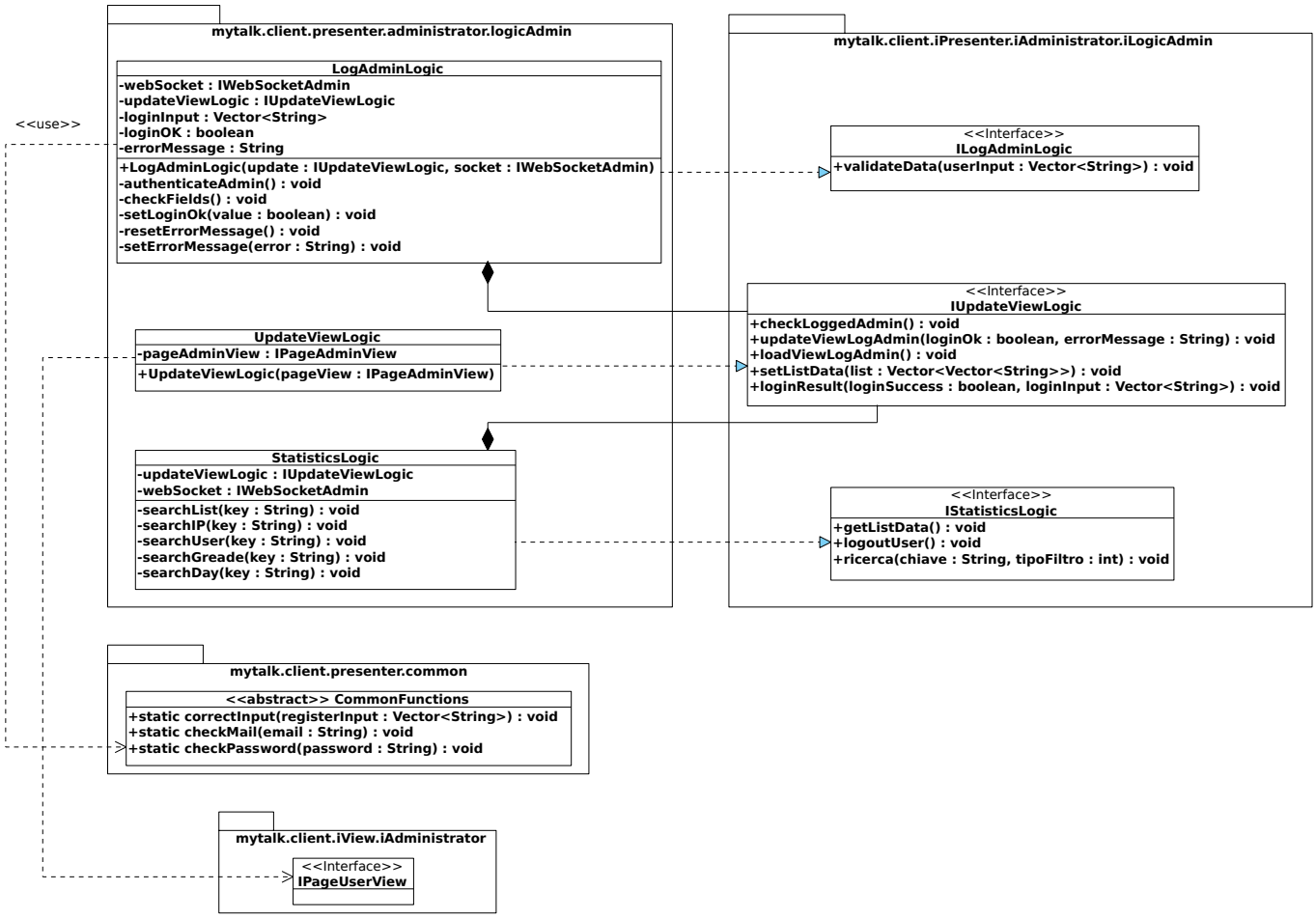

Figura 11: Diagramma delle classi dei package `mytalk.client.iPresenter.iAdministrator.iLogicAdmin` e `mytalk.client.presenter.administrator.logicAdmin`; dettaglio delle classi `ILogAdminLogic`, `IUpdateViewLogic`, `IStatisticsLogic`, `LogAdminLogic`, `UpdateViewLogic` e `StatisticsLogic`.

4.1.5.1 ILogAdminLogic

Funzione:
Interfaccia che offre operazioni logiche alle classi della View.

Relazioni con altre componenti:
L'interfaccia è implementata da:

- `mytalk.client.presenter.administrator.logicAdmin.LogAdminLogic`.

L'interfaccia è utilizzata da:

- `mytalk.client.view.administrator.LogAdminView`.

Metodi:
`+ void validateData(Vector<String> userInput);`
Controlla che il vettore `userInput` contenga valori conformi ai formati e richiede di controllare che tali valori siano anche presenti nel server$_{|g|}$.

4.1.5.2 IStatisticLogic

Funzione:
L'interfaccia fornisce i metodi tramite i quali l'utente amministratore può richiedere le statistiche al server$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.administrator.logicAdmin.StatisticLogic`.

L'interfaccia è utilizzata da:

`mytalk.client.view.user.StatisticView`;

Metodi:
`+ void getListData();`
Richiede al server$_{|g|}$ l'intera lista delle statistiche presenti nel database.

`+ void logoutUser();`
Richiede al server$_{|g|}$ l'uscita dal servizio da parte dell'utente amministratore attualmente autenticato.

`+ void ricerca(String chiave, int tipoFiltro);`
Richiede al server$_{|g|}$ un sottoinsieme delle statistiche, filtrate per un qualche valore.

`+ void setUserList();`
Richiede al server la lista di utenti registrati al servizio.

4.1.5.3 IUpdateViewLogic

Funzione:
Interfaccia che offre operazioni alle classi che devono interagire con la View.

Relazioni con altre componenti:
L'interfaccia è implementata da:

- `mytalk.client.view.administrator.UpdateViewLogic`.

L'interfaccia è utilizzata da:

- `mytalk.client.view.administrator.PageAdminView`.

Metodi:
`+ void checkLoggedAdmin();`
Controlla l'esistenza di cookies$_{|g|}$ di sessione.

`+ void updateViewLogAdmin(boolean loginOk, String errorMessage);`
Invia alla View il risultato dell'operazione di autenticazione.

`+ void loadViewLogAdmin();`
Richiede alla grafica il caricamento della GUI$_{|g|}$ di autenticazione dell'amministratore.

`+ void setListData(Vector<Vector<String>> list);`
Invia i dati delle statistiche alla View.

`+ void loginResult(boolean loginSuccess, Vector<String> loginInput);`
A seconda del valore di `loginSuccess` crea i cookie$_{|g|}$ o ritorna un'errore di autenticazione:

- se è `true`: crea il cookie$_{|g|}$ e richiede l'aggiornamento della View;
- se è `false`: imposta l'errore da ritornare e richiede l'aggiornamento della View.

`+ void errorViewFilter(String string);`
Richiede la visualizzazione dell'errore contenuto in `String`.

`+ void setUserList(Vector<String> listaUtenti);`
Invia alla View la lista degli utenti registrati.

`+ void removeCookies();`
Rimuove i cookie$_{|g|}$.

`+ void setUsernameLabel();`
Invia alla View il nome utente salvato nel cookie$_{|g|}$.

4.1.6 Package mytalk.client.presenter.administrator.logicAdmin

4.1.6.1 LogAdminLogic

Funzione:
La classe ha il compito di effettuare controlli di logica e di inviare i risultati ad altri metodi per il controllo o inviare i dati alla View per aggiornarla.

Relazioni con altre componenti:
Implementa l'interfaccia:

> `mytalk.client.iPresenter.iAdministrator.iLogicAdmin.ILogAdminLogic`.

Usa le classi:

> `mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic`;
>
> `mytalk.client.presenter.administrator.serverComUser.WebSocketAdmin`;

tramite le interfacce:

> `mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IUpdateViewLogic`;
>
> `mytalk.client.iPresenter.iAdministrator.iServerComUser.IWebSocketAdmin`.

Attributi:
- `IWebSocketAdmin webSocket`: riferimento alla classe `WebSocketAdmin`. Permette di richiamare i metodi per i controlli dei dati da parte del server.

- `IUpdateViewLogic updateViewLogic`: riferimento alla classe UpdateViewLogic. Permette di richiamare metodi per l'aggiornamento della $GUI_{|g|}$.

- `Vector<String> loginInput`: contenitore per i dati inseriti dall'utente in fase di autenticazione.

- `boolean loginOk`: valore $booleano_{|g|}$ che indica se il controllo del formato dei dati è andato a buon fine.

- `String errorMessage`: Stringa contenente il messaggio di errore complessivo.

Metodi:
`+LogAdminLogic(IUpdateViewLogic updateViewLogic, IWebSocketAdmin webSocket);`
Costruttore: inizializza gli oggetti `updateViewLogic` e `webSocket`. Imposta a `true` il valore di `loginOk`, inizializza il vettore `loginInput` e il messaggio

`errorMessage`.

`+ void validateData(Vector<String> userInput);`
Inserisce nel vettore `userInput` il parametro `userInput`, richiama il metodo per il controllo `checkField` e se il controllo è andato a buon fine richiama i metodi `resetErrorMessage()` e `authenticateAdmin()`. In caso contrario, richiama il metodo `updateViewLogAdmin(loginOk, errorMessage)` attraverso il riferimento `updateViewLogic`.

`- void authenticateAdmin();`
Richiama il metodo `authenticateAdmin(userInput)` attraverso il riferimento `webSocket`.

`- void checkFields();`
Controlla i valori di ingresso dell'utente e in caso di errore lo segnala impostando `loginOk` a `false` tramite il metodo `setLoginOk`.

`+ void setLoginOk(boolean loginOk);`
Imposta il valore di `loginOk` con il parametro `loginOk`.

`- void resetErrorMessage();`
Imposta al valore iniziale il messaggio di errore.

4.1.6.2 StatisticLogic

Funzione:
la classe ha il compito di richiedere al server$_{|g|}$ l'insieme delle statistiche memorizzate o un sottoinsieme di esse.

Relazioni con altre componenti:
La classe implementa l'interfaccia:

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IStatisticLogic`.

`mytalk.client.view.user.StatisticView`;
Utilizza le classi:

`mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic`;

`mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`;

tramite le interfacce:

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IUpdateViewLogic`;

`mytalk.client.iPresenter.iAdministrator.iServerComAdmin.IWebSocketAdmin`.

Attributi:
- `IWebSocketAdmin webSocket`: riferimento alla classe `WebSocketAdmin`. Permette di richiamare i metodi per l'autenticazione e la terminazione della sessione amministratore.

- `IUpdateViewLogic updateViewLogic`: riferimento alla classe `UpdateViewLogic`. Permette di richiamare i metodi di aggiornamento della View.

Metodi:
`+ StatisticLogic(IUpdateViewLogic updateViewLogic, IWebSocketAdmin webSocket);`
Costruttore a due parametri della classe.

`+ void getListData();`
Richiede al server$_{|g|}$ l'intera lista delle statistiche presenti nel database$_{|g|}$. Per far ciò invoca il metodo della classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin void getListData()`.

`+ void logoutUser();`
Richiede al server$_{|g|}$ l'uscita dal servizio da parte dell'utente amministratore attualmente autenticato.

`+ void ricerca(String chiave, int tipoFiltro);`
Richiede al server$_{|g|}$ un sottoinsieme delle statistiche, filtrate per un qualche valore.

`- void searchList(String key);`
Il metodo invia al server$_{|g|}$, tramite la classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`, un messaggio che richiede tutte le statistiche di chiamate riguardanti un determinato utente scelto tra quelli iscritti al server$_{|g|}$.

`- void searchIP(String key);`
Invia al server$_{|g|}$, tramite la classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`, un messaggio che richiede tutte le statistiche di chiamate riguardanti un determinato indirizzo IP$_{|g|}$.

`- void searchUser(String key);`
Invia al server$_{|g|}$, tramite la classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`, un messaggio che richiede tutte le statistiche di chiamate riguardanti un determinato nome utente, non scelto nella lista

degli utenti registrati al server$_{|g|}$, ma inserito tramite una form.

- `void searchGreade(String key);`
Invia al server$_{|g|}$, tramite la classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`, un messaggio che richiede tutte le statistiche di chiamate riguardanti un determinato indice di gradimento.

. `void searchDay(String key);`
Invia al server$_{|g|}$, tramite la classe `mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`, un messaggio che richiede tutte le statistiche di chiamate che si sono svolte in un determinato giorno.

+ `void setUserList();`
Richiede al server la lista di utenti registrati al servizio, per far ciò invia al server un messaggio XML$_{|g|}$ nella forma A.2.8.

+ `void setUsernameLabel();`
Richiama il metodo `setUsernameLabel()` della classe `mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic`.

4.1.6.3 UpdateViewLogic

Funzione:
La classe ha il compito di inviare i risultati delle operazioni del Presenter alla View.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.client.iPresenter.iAdministrator.iLogicAdmin.IUpdateViewLogic`.

Usa le classi:

`mytalk.client.view.administrator.PageAdminView`.

Tramite le interfacce:

`mytalk.client.iView.iAdministrator.IPageAdminView`.

Attributi:
- `IPageAdminView pageAdminView`: riferimento alla classe `PageAdminView`;

Metodi:
+ `UpdateViewLogic(IPageAdminView pageAdminView);`
Costruttore: inizializza l'oggetto pageAdminView;

`+ void checkLoggedAdmin();`
Recupera il nome dell'utente contenuto nel cookie$_{|g|}$ attraverso la chiamata `ManageCookies.getCookieUsername()`. Nel caso in cui il cookie$_{|g|}$ sia vuoto il metodo chiama `loadViewLogAdmin()`, altrimenti invoca `updateViewLogAdmin(true, ‘‘)`.

`+ void updateViewLogAdmin(boolean loginOk, String errorMessage);`
Il metodo ha lo scopo di aggiornare la schermata di login con eventuali errori riscontrati nell'inserimento delle credenziali, in alternativa se non sono stati riscontrati errori il sistema reindirizzerà l'utente nella schermata principale dell'applicazione. Richiama `updateViewLogAdmin(loginOk, errorMessage)` attraverso il riferimento `pageAdminView`.

`+ void loadViewLogAdmin();`
Richiama `loadViewLogAdmin()` attraverso il riferimento `pageAdminView`.

`+ void setListData(Vector<Vector<String>> list);`
Nel caso in cui `list` sia vuota richiama il metodo `errorViewFilter(\Non ci sono riscontri.)` e, in ogni caso, chiama il metodo `setListData(list)` attraverso il riferimento `pageAdminView`.

`+ void loginResult(boolean loginSuccess, Vector<String> loginInput);`
A seconda del valore di `loginSuccess` crea i cookie$_{|g|}$ o imposta un errore opportuno:

- se è `true`: crea il cookie$_{|g|}$ e richiama `updateViewLogAdmin(true,‘‘);`
- se è `false`: imposta l'errore e chiama `updateViewLogAdmin(false, errorMessage)`.

`+ void errorViewFilter(String error);`
Il metodo ha il compito di inviare alla view la richiesta di visualizzazione di un errore riscontrato nella ricerca di statistiche corrispondenti ad un determinato filtro. Per far ciò il metodo richiama `errorViewFilter(error)` attraverso il riferimento `pageAdminView`.

`+ void setUserList(Vector<String> listaUtenti);`
Il metodo ha il compito di inviare alla View la lista degli utenti registrati al servizio, per far ciò richiama `setUserList(listaUtenti)` attraverso il riferimento `pageAdminView`.

`+ void removeCookies();`
Richiede la rimozione dei cookie$_{|g|}$ attraverso `ManageCookies.deleteCookies()`.

`+ void setUsernameLabel();`
Richiama `setUsernameLabel(ManageCookies.getCookieUsername())` attraverso il riferimento `pageAdminView`, dove `ManageCookies.getCookieUsername()` recupera il nome dell'utente.

4.1.7 Package mytalk.client.iPresenter.iAdministrator.iServerComAdmin

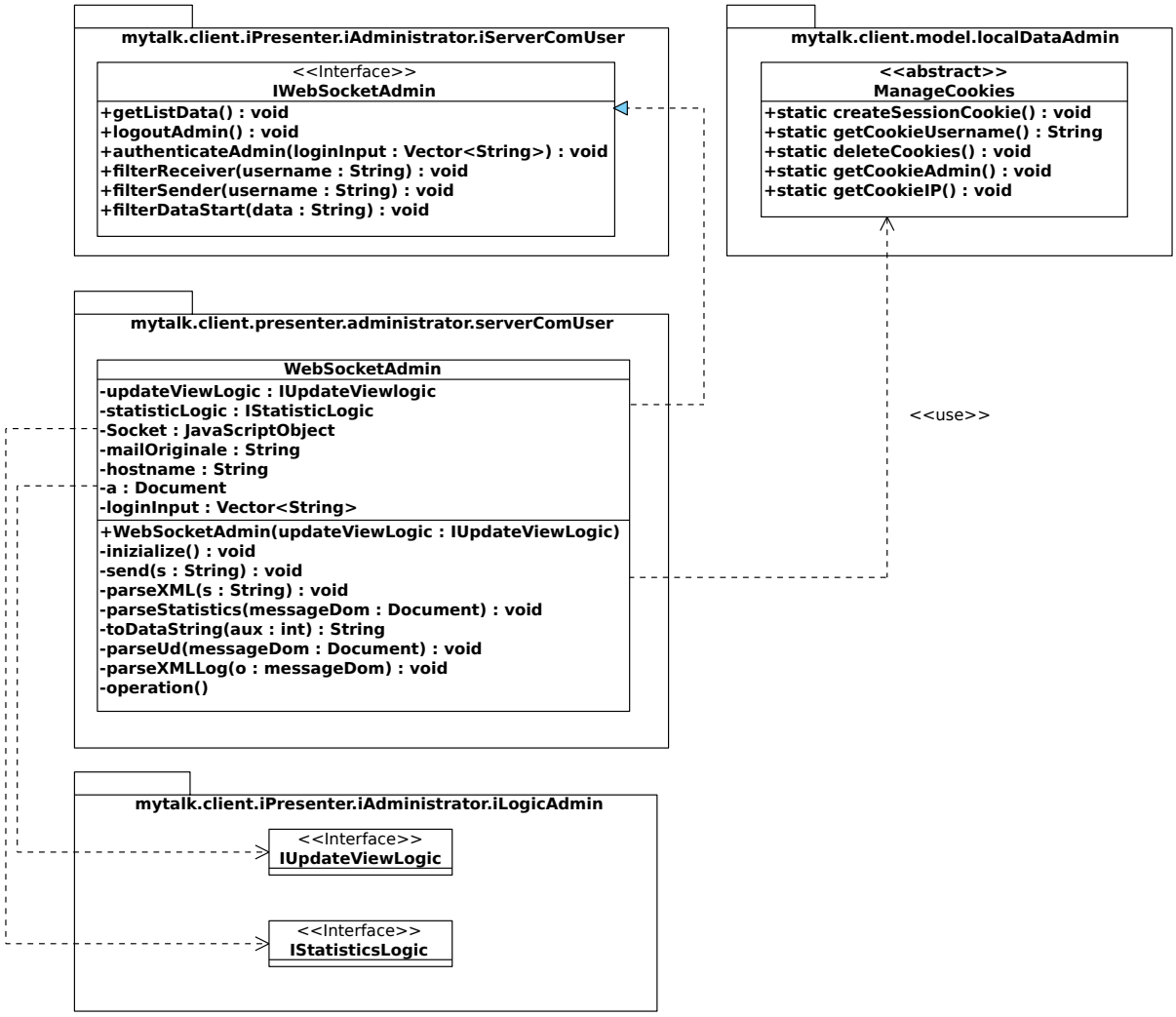

Figura 12: Diagramma delle classi dei package `mytalk.client.iPresenter.iAdministrator.iServerComAdmin` e `mytalk.client.presenter.administrator.serverComAdmin`; dettaglio delle classi `IWebSocketAdmin` e `WebSocketAdmin`.

4.1.7.1 IWebSocketAdmin

Funzione:
L'interfaccia fornisce i metodi tramite i quali è possibile richiedere al server$_{|g|}$ l'autenticazione di un amministratore e il reperimento delle statistiche.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`.

L'interfaccia è utilizzata da:

`mytalkadmin.client.presenter.logicAdmin.StatisticLogic`;

`mytalkadmin.client.presenter.logicAdmin.LogAdminLogic`.

Metodi:
`+ void getListData();`
Richiede l'insieme delle statistiche al server$_{|g|}$.

`+ void logoutAdmin(String data);`
Richiede l'uscita dal servizio da parte dell'amministratore.

`+ void authenticateAdmin(Vector<String> loginInput);`
Richiede l'autenticazione al servizio di amministratore.

`+ void filterSender(String username);`
Richiede le statistiche delle chiamate che coinvolgono un determinato utente chiamante.

`+ void filterDataStart(String data);`
Richiede le statistiche che hanno avuto una determinata data di inizio.

`+ filterGrade(String voto)`
Richiede al server$_{|g|}$ la lista della statistiche che hanno avuto un determinato giudizio da parte degli utenti.

`+ userList()`
Richiede al server$_{|g|}$ la lista degli utenti registrati al servizio.

`+ filterGrade(String voto)`
Richiede al server$_{|g|}$ la lista della statistiche che hanno avuto un determinato giudizio da parte degli utenti. Per far ciò invia un messaggio XML$_{|g|}$ al server nella forma A.2.12.

`- void inizialize()`
Inizializza i campi dati della classe e crea un oggetto di tipo `WebSocket` che permette di inviare e ricevere messaggi dal server$_{|g|}$.

`- void parseXMLUserList(Document messageDom)`
Viene richiamato per estrapolare nome utente, nome e cognome dalla lista di

utenti registrati al server$_{|g|}$ in formato XML$_{|g|}$ inviata dal server$_{|g|}$.
Una volta estratta la lista dal messaggio viene richiamato il metodo dell'oggetto `updateViewLogic setUserList(Vector<String> utenti)` per inviare la lista alla View.

4.1.8 Package mytalk.client.presenter.administrator.serverComAdmin

Il diagramma delle classi è contenuto nell'immagine 4.1.5.

4.1.8.1 WebSocketAdmin

Funzione:
la classe ha il compito di gestire la comunicazione con il server$_{|g|}$ e di richiamare gli opportuni metodi della classe `UpdateViewLogic` per aggiornare l'interfaccia grafica.

Relazioni con altre componenti:
La classe implementa l'interfaccia:

> `mytalk.client.iPresenter.iAdministrator.iServerComAdmin.IWebSocketAdmin`.

La classe utilizza la classe:

> `mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic;;`

Attributi:
- `IUpdateViewLogic updateViewLogic`: riferimento all'oggetto che ha il compito di aggiornare l'interfaccia GUI$_{|g|}$.

- `JavaScriptObject Socket`: riferimento all'oggetto che rappresenta il WebSocket e che ha la funzione di inviare e ricevere messaggi dal server$_{|g|}$.

- `static String hostname`: stringa che indica il nome dell'host sul quale risiede il WebSocket. Il valore è - `ws://IP-address:8080/MyTalk-server/WSAdmin`.

- `Document a;`: il tipo Document serve a contenere e verificare la buona formazione della stringa XML$_{|g|}$ inviata dal server$_{|g|}$.

- `Vector<String> loginInput`: il vettore contiene i dati con i quali l'utente amministratore si è autenticato (o i dati con cui ha tentato di farlo).

Metodi:
`+ WebSocketAdmin(IUpdateViewLogic updateViewLogic)`

Costruttore ad un parametro: richiama il metodo `inizialize()` dopo aver assegnato il riferimento passatogli come parametro al campo dati `updateViewLogic`.

`+ void getListData();`
Il metodo ha lo scopo di richiedere l'insieme delle statistiche al server$_{|g|}$, tramite un messaggio XML$_{|g|}$ nella forma A.2.12.

`+ void logoutAdmin(String data);`
Il metodo ha il compito di richiedere l'uscita dal servizio da parte dell'amministratore, tramite un messaggio XML$_{|g|}$ nella forma A.2.3.

`+ void authenticateAdmin(Vector<String> loginInput);`
Il metodo richiede l'autenticazione al servizio di amministratore, tramite un messaggio XML$_{|g|}$ nella forma A.2.2.

`+ void filterSender(String username);`
Il metodo richiede le statistiche delle chiamate che coinvolgono un determinato utente chiamante, tramite un messaggio XML$_{|g|}$ nella forma A.2.12.

`+ void filterDataStart(String data);`
Il metodo richiede le statistiche che hanno avuto una determinata data di inizio, tramite un messaggio XML$_{|g|}$ nella forma A.2.12.

`- void parseXML(String XML`$_{|g|}$ `);`
Il metodo riceve come parametro una stringa XML$_{|g|}$, ne controlla la buona formazione e il contenuto e a seconda di quest'ultimo richiama l'opportuno metodo.

`- void parseStatistics(Document messageDom);`
Il metodo viene richiamato se il messaggio XML$_{|g|}$ proveniente dal server$_{|g|}$ contiene delle statistiche. Il metodo prima estrae i valori delle varie statistiche dal messaggio e poi richiama il metodo della classe `UpdateViewLogic setListData(Vector<Vector<String>> stats)` che invierà i dati all'interfaccia grafica così che possano essere visualizzati.

`- void parseUd(Document messageDom);`
Il metodo viene richiamato se il messaggio XML$_{|g|}$ proveniente dal server$_{|g|}$ contiene delle informazioni riguardanti l'autenticazione dell'utente amministratore.

`- void parseXMLLog(Document messageDom);`
Il metodo viene richiamato se si riceve un responso da parte del server$_{|g|}$ su una richiesta di autenticazione precedentemente inviata dall'utente amministratore.

`+ userList();`
Richiede al server$_{|g|}$ la lista degli utenti registrati al servizio. Per far ciò invia

al server un messaggio nella forma A.2.8.

4.1.9 Package mytalk.client.presenter.user.logicUser.common

Il diagramma delle classi è contenuto nell'immagine 4.1.1.

4.1.9.1 CommonFunctions

Funzione:
Classe astratta che contiene metodi di utilità per il controllo delle stringhe che vengono utilizzati in più classi.

Relazioni con altre componenti:
I suoi metodi vengono richiamati dalle classi:

`mytalk.client.presenter.user.logicUser.LogUserLogic`;

`mytalk.client.presenter.user.logicUser.RegisterLogic`;

`mytalk.client.presenter.user.logicUser.DataUserLogic`.

Attributi:
Nessuno.

Metodi:
`+ static boolean checkEmail(String email);`
Controlla che la mail sia ben formata. Ritorna l'esito di tale controllo.

`+ static boolean ckechPassword(String password);`
Controlla che la password sia di almeno 8 caratteri. Ritorna l'esito di tale controllo.

`+ static boolean checkString(String n);`
Controlla che il parametro ricevuto corrisponda a un nome o un cognome, cioè che sia solo formato da lettere e spazi. Ritorna l'esito di tale controllo.

`+ static boolean checkPasswordControl(String psw, String pswC);`
Controlla che i due parametri `psw` e `pswC` siano uguali. Ritorna l'esito di tale controllo.

`+ static boolean checkCompany(String company);`
Controlla che il parametro ricevuto, che corrisponde al nome dell'azienda con il quale l'utente intende registrarsi, non sia formato solo da numeri. Ritorna l'esito di tale controllo.

`+ static boolean checkNumber(String number);`
Controlla che il parametro ricevuto sia un numero telefonico nel formato corretto. Ritorna l'esito di tale controllo.

`+ static void correctInput(Vector<String> userInput);`
Effettua delle correzioni di base sull'input dell'utente: toglie eventuali spazi inutili all'inizio e alla fine delle stringhe e converte l'indirizzo e-mail in minuscolo.

4.1.10 Package mytalk.client.iPresenter.iUser.iServerComUser

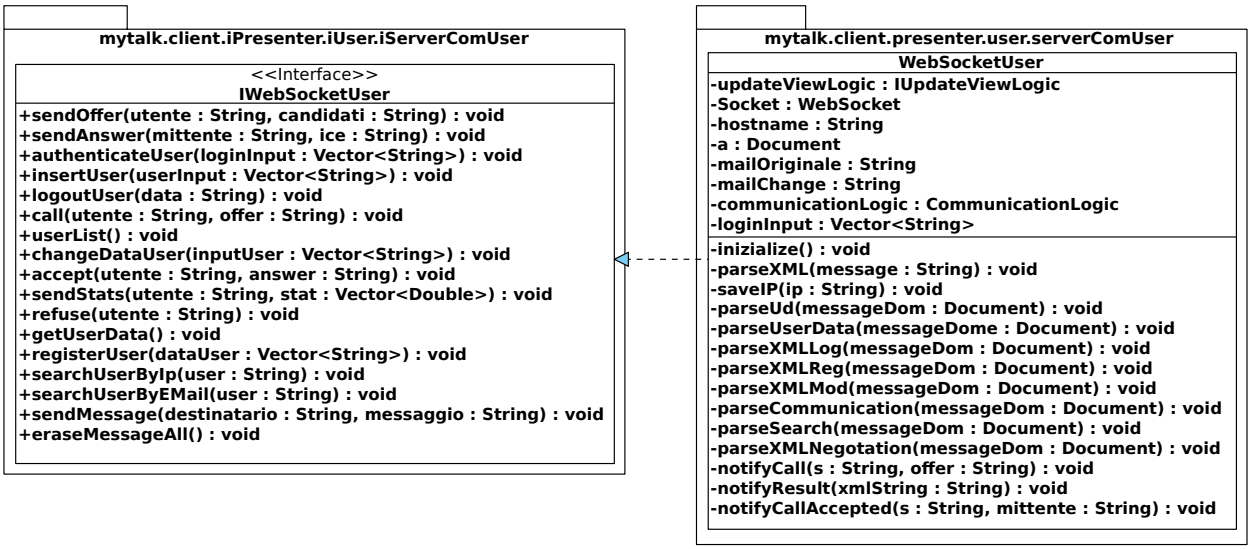

Figura 13: Diagramma delle classi dei package `mytalk.client.iPresenter.iUser.iServerComUser` e `mytalk.client.presenter.user.serverComUser`; dettaglio delle classi `IWebSocketUser` e `WebSocketUser`.

4.1.10.1 IWebSocketUser

Funzione: le classi che implementano questa interfaccia hanno il compito di comporre ed inviare i messaggi verso il server$_{|g|}$e di ricevere e tradurre i messaggi provenienti dal server$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata dalla classe:

`mytalk.client.presenter.user.serverComUser.WebSocketUser`.

L'interfaccia è utilizzata da:

`mytalk.client.presenter.user.logicUser.LogUserLogic`;

`mytalk.client.presenter.user.logicUser.RegisterLogic`;

`mytalk.client.presenter.user.logicUser.DataUserLogic`;

`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Metodi:
`+ void accept(String utente, String answer);`
Invia un messaggio al server$_{|g|}$ che poi verrà recapitato all'utente ricevente se questo è online. Il messaggio indica che l'utente mittente ha accettato la chiamata dell'utente ricevente e fornisce all'utente che ha chiamato la descrizione della sessione locale dell'utente.

`+ void call(String utente, String offer);`
Invia un messaggio all'utente passato come parametro, il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente, il messaggio indica che l'utente mittente vuole instaurare una comunicazione con l'utente ricevente e per far ciò fornisce all'utente remoto la descrizione della sessione locale.

`+ void refuse(String utente);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio indica che l'utente mittente ha rifiutato la richiesta di comunicazione dell'utente ricevente.

`+ void sendOffer(String utente, String iceCandidate);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio contiene inoltre gli IceCandidate della sessione dell'utente.

`+ void sendAnswer(String mittente, JavaScriptObject description);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio contiene inoltre gli IceCandidate della sessione dell'utente.

`+ void changeDataUser(Vector<String> inputUser);`
Invia al server$_{|g|}$ i nuovi dati personali dell'utente.

`+ void sendStats(String ricevente, Vector<Double> stats);`
Il metodo, invocato al termine della chiamata invia le statistiche al server$_{|g|}$. Quest'ultimo le memorizza nella base di dati per renderle disponibili agli utenti amministratori.

`+ void authenticateUser(Vector<String> loginInput);`
Richiede al server$_{|g|}$ l'autenticazione dell'utente i cui dati di login sono passati come parametro.

`+ void logoutUser(String data);`
Richiede al server$_{|g|}$ l'uscita dal servizio dell'utente i cui dati di login sono stati passati come parametro.

`+ void userList();`
Richiede al server$_{|g|}$ la lista degli utenti registrati al servizio.

`+ void getUserData();`
Richiede al server$_{|g|}$ i dati personali dell'utente autenticato.

`+ void insertUser(Vector<String> dataUser);`
Il metodo ha lo scopo di inviare al server$_{|g|}$ la richiesta di registrazione di un nuovo utente al servizio.

`+ void searchUserByIP(String ip);`
Il metodo richiede al server di cercare un utente che abbia associato un determinato indirizzo IP$_{|g|}$.

`+ void searchUserByEmail(String username);`
Il metodo richiede al server di cercare un utente che abbia associato una determinata e-mail.

`+ void sendMessage(String destinatario, String message);`
Invia un messaggio all'utente indicato dalla stringa `destinatario`.

`+ void eraseMessageAll();`
Elimina tutti i messaggi della casella di posta dell'utente.

4.1.11 Package mytalk.client.presenter.user.serverComUser

4.1.11.1 WebSocketUser

Funzione:
La classe ha il compito di comporre ed inviare i messaggi verso il server$_{|g|}$. Riceve e traduce i messaggi provenienti dal server$_{|g|}$.

Relazioni con altre componenti:
La classe implementa l'interfaccia:

- `mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`.

La classe è utilizzata da:

`mytalk.client.presenter.user.logicUser.LogUserLogic`;
`mytalk.client.presenter.user.logicUser.RegisterLogic`;
`mytalk.client.presenter.user.logicUser.DataUserLogic`;
`mytalk.client.presenter.user.logicUser.CommunicationLogic`.

Tramite l'interfaccia:

- `mytalk.client.iPresenter.iUser.iServerComUser.IWebSocketUser`.

Attributi:
`- IUpdateViewLogic updateViewLogic`: riferimento all'oggetto che gestisce gli aggiornamenti della View.

- `JavaScriptObject socket`: riferimento all'oggetto che rappresenta il WebSocket.

- `String hostname`: indirizzo del WebSocket sulla componente server$_{|g|}$.

- `Document a`: documento XML$_{|g|}$. Tramite questo oggetto viene controllato che il messaggio XML$_{|g|}$ proveniente dal server$_{|g|}$ sia ben formato.

- `String MailOriginale`: contiene l'e-mail con la quale l'utente si è autenticato.

- `String MailChange`: contiene l'e-mail la nuova e-mail impostata dall'utente. Se l'utente non ha cambiato l'e-mail in questa sessione il campo è uguale a `MailOriginale`.

- `ICommunicationLogic communicationLogic`: riferimento all'oggetto che gestisce la comunicazione.

- `Vector<String> loginInput`: contiene i dati con i quali l'utente si è autenticato o ha tentato di farlo.

Metodi:

`+ WebSocketUser(IUpdateViewLogic updateViewLogic);`
Costruttore: riceve come parametro un oggetto di tipo `mytalk.client.presenter.user.logicUser.UpdateViewLogic` e con esso inizializza il campo dati `updateViewLogic`.

`+ public void inizialize();`
Inizializza il campo dati `webSocket` e ne registra gli eventi.

`+ void accept(String utente, String answer);`
Invia un messaggio al server$_{|g|}$ che poi verrà recapitato all'utente ricevente se questo è online. Il messaggio indica che l'utente mittente ha accettato la chiamata dell'utente ricevente e fornisce al chiamante la descrizione della sessione locale di tipo `answer` dell'utente codificata in formato JSON$_{|g|}$.

`+ void call(String utente, String offer);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio indica che l'utente mittente vuole instaurare una comunicazione con l'utente ricevente e per far ciò fornisce all'utente remoto la descrizione della sessione locale di tipo `offer` codificata in formato JSON.

`+ void refuse(String utente);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene

prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio indica che l'utente mittente ha rifiutato la richiesta di comunicazione dell'utente ricevente.

`+ void sendOffer(String utente, String iceCandidate);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio contiene inoltre gli IceCandidate della sessione dell'utente.

`+ void sendAnswer(String mittente, String iceCandidate);`
Invia un messaggio all'utente passato come parametro. Il messaggio viene prima inviato al server$_{|g|}$ e poi da questo viene inviato all'utente ricevente. Tale messaggio contiene inoltre gli IceCandidate della sessione dell'utente.

`+ void changeDataUser(Vector<String> inputUser);` Invia al server$_{|g|}$ i nuovi dati personali dell'utente.

`+ void sendStats(String ricevente, Vector<Double> stats);`
Il metodo, invocato al termine della chiamata, invia le statistiche al server$_{|g|}$ che poi le memorizzerà nella base di dati per renderle disponibili agli utenti amministratori.

`+ void authenticateUser(Vector<String> loginInput);`
Richiede al server$_{|g|}$ l'autenticazione dell'utente i cui dati di login sono passati come parametro.

`+ void logoutUser(String data);`
Richiede al server$_{|g|}$ l'uscita dal servizio dell'utente i cui dati di login sono stati passati come parametro.

`+ void userList();`
Richiede al server$_{|g|}$ la lista degli utenti registrati al servizio.

`+ void getUserData();`
Richiede al server$_{|g|}$ i dati personali dell'utente autenticato.

`- void parseXML(String s);`
Esegue il parsing di una stringa XML$_{|g|}$ giunta dal server$_{|g|}$ tramite WebSocket, identificandone il tipo di messaggio dal nodo radice. Il messaggio può essere di tre tipi:

1. `ip`: riceve l'indirizzo IP$_{|g|}$ dal server$_{|g||g|}$.
2. `userData`: risponde ad una richiesta di login o di registrazione o di modifica dei dati dell'utente che ha inviato la richiesta.
3. `communication`: esegue la negoziazione della chiamata e l'interscambio delle sessioni remote.

Il metodo, a seconda del contenuto del nodo radice, richiama un opportuno metodo che avrà il compito di estrarre le informazioni dal messaggio XML$_{|g|}$.

- `void saveIP(String s);`
Viene invocato qualora dal server$_{|g|}$ provenga un messaggio contenente l'indirizzo IP$_{|g|}$ dell'utente. Viene invocato il metodo della classe `ManageCookies` per salvare il dato in un cookie$_{|g|}$.

- `void parseUd(Document messageDom);`
Viene invocato dal metodo `parseXML` se il messaggio proveniente dal server$_{|g|}$ contiene informazioni riguardanti:
  - login: viene richiamato l'opportuno metodo `parseXMLLog(Document messageDom);`
  - registrazione: viene richiamato il metodo `parseXMLReg(Document messageDom);`
  - modifica dei dati personali dell'utente: viene richiamato il metodo `parseXMLMod(Document messageDom);`
  - dati personali dell'utente: viene richiamato il metodo `parseUserData(Document messageDom);`

- `void parseXMLLog(Document o);`
Estrapola l'esito di una richiesta di autenticazione avvenuta in precedenza e richiama il metodo dell'oggetto `updateViewLogic loginResult(boolean loginSucces, Vector<String> loginInput)` per notificare l'esito all'utente.

- `void parseXMLReg(Document o);`
Estrapola l'esito di una richiesta di registrazione avvenuta in precedenza e richiama il metodo dell'oggetto `updateViewLogic registerResult(boolean registerSuccess` per notificare l'esito all'utente.

- `void parseXMLMod(Document o);`
Estrapola l'esito di una richiesta di modifica dei dati personali dell'utente avvenuta in precedenza e richiama il metodo dell'oggetto `updateViewLogic resultDataUser(boolean update)` per notificare l'esito all'utente.

- `void parseUserData(Document o);`
Estrapola i dati personali dell'utente dal documento XML$_{|g|}$ passato come parametro. I dati vengono quindi passati al metodo `setUserDataLabel(Vector<String> dataUser)` dell'oggetto `updateViewLogic`.

- `void parseCommunication(Document messageDom);`
Viene invocato dal metodo `parseXML` se il messaggio proveniente dal server$_{|g|}$ contiene informazioni riguardanti la parte relativa alla comunicazione. In particolare:

- negoziazione: quando un utente tramite il server$_{|g|}$ invia un messaggio all'utente che può indicare una chiamata, l'accettazione di una chiamata o il rifiuto di una chiamata. In questi casi viene chiamato il metodo `parseXMLNegotation(Document messageDom)`.
- scambio: quando la chiamata è stata già accettata dall'utente chiamato e l'allacciamento della comunicazione è già in fase avanzato. Questa fase riguarda lo scambio degli IceCandidate e se il messaggio contiene uno o più di questi oggetti viene richiamato il metodo `parseXMLExchange(Document messageDom)`.
- ricerca: contiene il responso che il server$_{|g|}$ fornisce in seguito alla richiesta di un utente di chiamare o digitando il nome dell'utente ricevente o il suo indirizzo IP$_{|g|}$ e non di selezionarlo dalla lista di quelli registrati al server$_{|g|}$. Per estrapolare l'esito della ricerca viene richiamato il metodo - `void parseSearch(Document messageDom)`.
- richiesta lista utenti: contiene la lista degli utenti registrati al server$_{|g|}$. Per estrapolare l'esito della ricerca viene richiamato il metodo `parseXMLUserList(Document messageDom)`.

- `void parseSearch(Document messageDom);`
Estrapola il nome utente fornito dal server$_{|g|}$ o notifica l'esito negativo della ricerca alla componente grafica. Se la ricerca ha avuto successo allora viene chiamato l'utente corrispondente tramite il metodo `call(String utente)` dell'oggetto `communicationLogic`.

- `void parseXMLNegotation(Document r);`
A seconda che si tratti di un'offerta (una chiamata in entrata) o di una risposta (l'indicazione dell'accettazione o meno della comunicazione da parte dell'altro utente) richiama il metodo `notifyCall(String user, String offer)` se si tratta di una chiamata in entrata o `notifyResult(String user)` se si tratta del responso di una precedente richiesta.

- `void notifyCall(String s, String offer);`
Viene richiamato per notificare una chiamata in arrivo all'utente ed, eventualmente, per impostare la descrizione remota proveniente dall'utente chiamante. Richiama i metodi `callEnter(String offer)` dell'oggetto `communicationLogic` e `notifyCall(String utente)` dell'oggetto `updateViewLogic`.

- `void notifyResult(Document messageDom);`
Estrapola l'accettazione o il rifiuto dell'utente chiamato dal messaggio XML$_{|g|}$ proveniente dal server$_{|g|}$. Se il messaggio è di accettazione viene estratta anche la descrizione remota di tipo `answer`, che deve essere quindi presente e viene richiamato il metodo `notifyCallAccepted(String answer, String mittente)` al quale vengono inviati come parametri la descrizione di sessione remota e il nome dell'utente che ha inviato il messaggio. Se il messaggio è di rifiuto viene richiamato il metodo `notifyCallRefused(String mittente)`

al quale viene passato come parametro il nome dell'utente che ha inviato il messaggio.

`- void notifyCallAccepted(String answer, String mittente);`
Notifica alla classe `CommunicationLogic` l'accettazione della chiamata da parte dell'utente chiamato. Per la notifica viene richiamato il metodo `acceptedCall(answer, mittente)`.

`- void notifyCallRefused(String mittente);`
Notifica alla classe `UpdateViewLogic` il rifiuto della chiamata da parte dell'utente chiamato. Per la notifica viene richiamato il metodo `refusedCall(mittente)`.

`- void parseXMLUserList(Document messageDom);`
Il metodo viene richiamato per estrapolare il nome utente, nome e cognome dalla lista di utenti registrati al server$_{|g|}$ in formato xml$_{|g|}$ inviata dal server$_{|g|}$. Una volta estratta la lista dal messaggio viene richiamato il metodo dell'oggetto `updateViewLogic setUserList(Vector<String> utenti)` per inviare la lista alla View.

`- void parseXMLExchange(Document messageDom);`
Il metodo viene richiamato quando vengono ricevuti gli `IceCandidate` da parte dell'altro utente. Gli `IceCandidate` vengono estratti dal messaggio proveniente dal server$_{|g|}$ e se l'utente che gli ha inviati è quello che ha iniziato la chiamata, viene richiamato il metodo `receivedOffer(String mittente, Vector<String> iceCandidate)`. Se invece l'utente che ha inviato il messaggio è quello che ha accettato la chiamata, viene richiamato il metodo `receivedResponse(iceCandidate)`.

`- void receivedOffer(String mittente, Vector<String> candidate);`
Notifica il ricevimento degli IceCandidate alla classe `CommunicationLogic` richiamando il metodo `receivedSession(String mittente, Vector<String> candidate)` dell'oggetto `communicationLogic`.

`- void receivedResponse(String mittente, Vector<String> candidate);`
Il metodo notifica il ricevimento degli IceCandidate alla classe `CommunicationLogic` richiamando il metodo `receivedSessionPerformed(Vector<String> candidate)` dell'oggetto `communicationLogic`.

`+ void insertUser(Vector<String> dataUser);`
Invia al server la richiesta di registrazione di un nuovo utente al servizio, per far ciò invia al server$_{|g|}$ un messaggio XML$_{|g|}$ nella forma A.2.4 utilizzando le stringhe presenti nel vettore passato come parametro.

`+ void searchUserByIP(String ip);`
Richiede al server$_{|g|}$ di cercare un utente che abbia associato un determinato indirizzo IP$_{|g|}$. Per far ciò invia un messaggio XML$_{|g|}$ al server$_{|g|}$ nella forma A.2.11.

`+ void searchUserByEmail(String username);`
Richiede al server di cercare un utente che abbia associato un determinato indirizzo e-mail. Per far ciò invia al server$_{|g|}$ un messaggio XML$_{|g|}$ nella forma A.2.11.

`+ void sendMessage(String destinatario, String message);`
Invia, grazie al server, un messaggio all'utente indicato dalla stringa `destinatario`.

`+ void eraseMessageAll();`
Elimina, grazie al server, tutti i messaggi della casella di posta dell'utente.

4.2 Server

La sotto-componente Server$_{|g|}$ gestisce le comunicazioni ricevute ed inviate dalla sotto-componente Client$_{|g|}$ (vedi 4.1). A questo scopo interroga la sotto-componente Server$_{|g|}$ del Model (vedi 5.2) e compone i messaggi XML$_{|g|}$ risultato (vedi A).
È composta dalle classi:

`mytalk.server.presenter.XMLField` (sez. 4.2.1.1)

`mytalk.server.presenter.administrator.logicAdmin.IWSAdmin` (sez. 4.2.2.1)

`mytalk.server.presenter.administrator.logicAdmin.IManageWSA` (sez. 4.2.2.2)

`mytalk.server.presenter.administrator.logicAdmin.WSAdmin` (sez. 4.2.2.3)

`mytalk.server.presenter.administrator.logicAdmin.WSAdmin.WebSocket` (sez. 4.2.2.4)

`mytalk.server.presenter.administrator.logicAdmin.ManageWSA` (sez. 4.2.2.5)

`mytalk.server.presenter.user.logicUser.IWSUser` (sez. 4.2.3.1)

`mytalk.server.presenter.user.logicUser.IManageWSU` (sez. 4.2.3.2)

`mytalk.server.presenter.user.logicUser.WSUser` (sez. 4.2.3.3)

`mytalk.server.presenter.user.logicUser.WSUser.WebSocket` (sez. 4.2.3.4)

`mytalk.server.presenter.user.logicUser.ManageWSU` (sez. 4.2.3.5)

4.2.1 Package mytalk.server.presenter

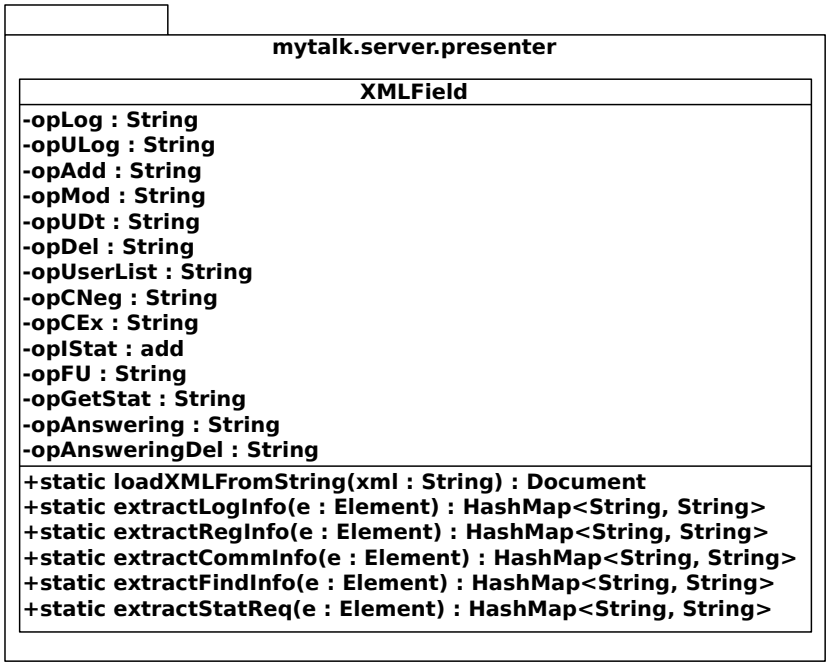

Figura 14: Diagramma delle classi del package `mytalk.server.presenter`; dettaglio della classe `XMLField`.

4.2.1.1 XMLField

Funzione:
La classe ha lo scopo di raccogliere tutti i metodi e gli attributi utili per l'estrazione delle informazioni nei messaggi in formato $\text{XML}_{|g|}$ pervenuti dal $\text{client}_{|g|}$. Le informazioni estrapolate vengono convertite in un formato gestibile dalle classi del package `mytalk.server.model`.

Relazioni con altre componenti:
Usa le classi:

`mytalk.server.model.dao.DataAccessObject`;

`mytalk.server.model.dao.ObjectTransfer`.

tramite l'interfaccia:

`mytalk.server.model.dao.IDataAccessObject`;

`mytalk.server.model.dao.IObjectTransfer`;

Attributi:
`+ static final String opLog`: identificativo stringa dell'operazione di login nei messaggi $\text{XML}_{|g|}$.

`+ static final String opULog`: identificativo stringa dell'operazione di logout nei messaggi $XML_{|g|}$.

`+ static final String opAdd`: identificativo stringa dell'operazione di aggiunta nei messaggi $XML_{|g|}$.

`+ static final String opMod`: identificativo stringa dell'operazione di modifica nei messaggi $XML_{|g|}$.

`+ static final String opUDt`: identificativo stringa dell'operazione per ottenere i dati utente nei messaggi $XML_{|g|}$.

`+ static final String opDel`: identificativo stringa dell'operazione di eliminazione nei messaggi $XML_{|g|}$.

`+ static final String opUserList`: identificativo stringa dell'operazione per ottenere la lista degli utenti nei messaggi $XML_{|g|}$.

`+ static final String opCNeg`: identificativo stringa dell'operazione di negoziazione della comunicazione nei messaggi $XML_{|g|}$.

`+ static final String opCEx`: identificativo stringa dell'operazione di scambio dati della comunicazione nei messaggi $XML_{|g|}$.

`+ static final String opIStat` : identificativo stringa dell'operazione per l'inserimento dati statistici delle comunicazioni nei messaggi $XML_{|g|}$.

`+ static final String opFU`: identificativo stringa dell'operazione di ricerca utente nei messaggi $XML_{|g|}$.

`+ static final String opGetStat`: identificativo stringa dell'operazione di richiesta informazioni statistici.

`+ static final String opAnswering`: identificativo stringa dell'operazione di inserimento nuovo messaggio di segreteria.

`+ static final String opAnsweringDelete`: identificativo stringa dell'operazione di rimozione dei messaggi in segreteria.

`+ static final HashMap<String, String> reference`: raccolta di riferimenti per i nodi $XML_{|g|}$.

Metodi:

```
+ static Document loadXMLFromString(String xml) throws Exception;
```

Gestisce la lettura della stringa XML$_{|g|}$ (data dal parametro `xml`) e la converte in una struttura dati `Document` che facilita, per la VM Java$_{|g|}$, la lettura delle informazioni dei nodi e la manipolazione di questi. Se la lettura della stringa XML$_{|g|}$ fallisce, il metodo solleva una generica eccezione `Exception`.

`+ static HashMap<String, String> extractLogInfo(Element e);`
Estrapola i dati di login dell'utente e ne restituisce i valori associati. Il parametro `e` è una struttura dati contenente i valori estratti dalla stringa XML$_{|g|}$ originaria. La tabella Hash di ritorno contiene i valori, associati a riferimenti specifici, utili per la gestione e la verifica dei dati di login.

`+ static HashMap<String, String> extractRegInfo(Element e);`
Estrapola i dati di registrazione dell'utente e ne restituisce i valori associati. Il parametro `e` è una struttura dati contenente i valori estratti dalla stringa XML$_{|g|}$ originaria. La tabella Hash di ritorno contiene i valori, associati a riferimenti specifici, utili per la gestione e verifica dei dati di registrazione.

`+ static HashMap<String, String> extractCommInfo(Element e);`
Estrapola i dati statistici della comunicazione avvenuta tra gli utenti e ne restituisce i valori associati. Il parametro in ingresso `e` è una struttura dati contenente i valori estratti dalla stringa XML$_{|g|}$ originaria. La tabella Hash di ritorno contiene i valori, associati a riferimenti specifici, utili per la gestione e verifica dei dati statistici della comunicazione.

`+ static HashMap<String, String> extractFindInfo(Element e);`
Estrapola i dati di ricerca dell'utente e ne restituisce i valori associati. Il parametro in ingresso `e` è una struttura dati contenente i valori estratti dalla stringa XML$_{|g|}$ originaria. La tabella Hash di ritorno contiene i valori, associati a riferimenti specifici, utili per la gestione e verifica dei dati di ricerca.

`+ static HashMap<String, String> extractStatReq(Element e);`
Estrapola i dati di ricerca per ottenere i valori statistici richiesti.
Il parametro in ingresso `e` è una struttura dati contenente i valori estratti dalla stringa XML$_{|g|}$ originaria. La tabella Hash di ritorno contiene i valori, associati a riferimenti specifici, utili per la gestione e verifica dei dati di ricerca.

4.2.2 Package mytalk.server.presenter.administrator.logicAdmin

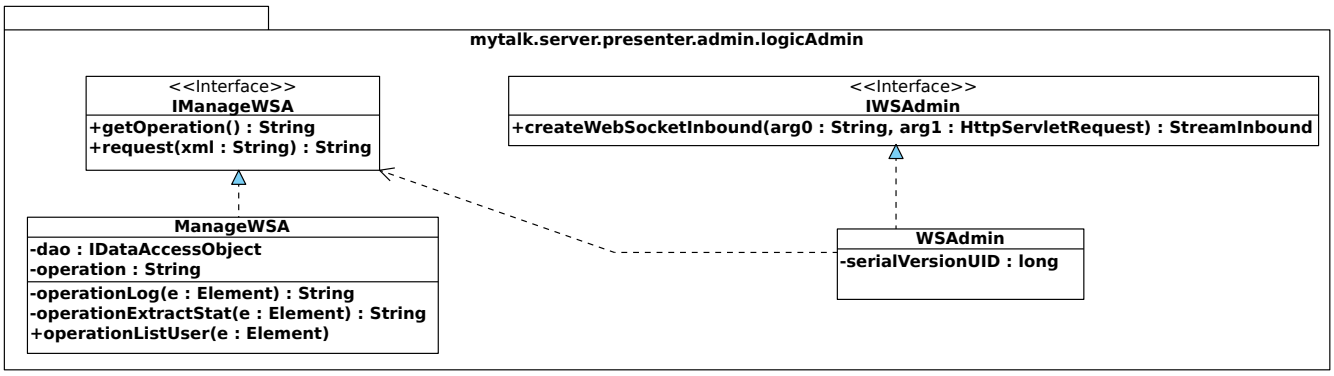

Figura 15: Diagramma delle classi del package `mytalk.server.presenter.administrator.logicAdmin`; dettaglio delle classi `IWSAdmin`, `IManageWSA`, `WSAdmin` e `ManageWSA`.

4.2.2.1 IWSAdmin

Funzione:
Interfaccia che gestisce le connessioni da parte dei Client$_{|g|}$ di tipo amministratore e mantiene attivo il riferimento per l'ingresso e l'uscita di messaggi da/per il client$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.presenter.administrator.logicAdmin.WSAdmin`.

Metodi:
`+ createWebSocketInbound(String arg0, HttpServletRequest arg1);`
Ad ogni sua invocazione, crea l'oggetto `WSAdmin.WebSocket`.

4.2.2.2 IManageWSA

Funzione:
Interfaccia che gestisce i messaggi inviati dall'utente amministratore elaborandoli ed estrapolandone le informazioni di ricerca per poi restituire un messaggio, correttamente composto, al medesimo utente.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.presenter.administrator.logicAdmin.ManageWSA`.

Metodi:
`+ String getOperation();`
Ritorna il nome dell'ultima elaborazione effettuata dall'oggetto.

`+ String request(String xml);`
Elabora la stringa in ingresso selezionando l'operazione associata da eseguire (quella richiesta dal messaggio) restituendo il messaggio risultante.

4.2.2.3 WSAdmin

Funzione:
La classe serve da riferimento per le comunicazioni client-server$_{|g|}$ per gli utenti di tipo amministratore. Deve inoltre creare l'oggetto `logicAdmin.WebSocket` per l'invio e ricezione delle comunicazioni.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.presenter.administrator.logicAdmin.IWSAdmin`.

Usa le classi:

`mytalk.server.presenter.administrator.logicAdmin.ManageWSA`;

tramite l'interfaccia:

`mytalk.server.presenter.administrator.logicAdmin.IManageWSA`.

Crea la classe:

`mytalk.server.presenter.administrator.logicAdmin.WSAdmin.WebSocket`.

Attributi:
`- static final long serialVersionUID`: Valore per la classe seriale.

Metodi:
`+ StreamInbound createWebSocketInbound(String arg0, HttpServletRequest arg1);`
Gestisce le richieste di connessione pervenute dai client$_{|g|}$ amministratori. Il parametro `arg0` serve a specificare un sub-protocollo di comunicazione tra client$_{|g|}$ e server$_{|g|}$. Se non specificato, il suo valore è `null`. Il parametro `arg1` serve a riferire la richiesta HTTP$_{|g|}$ catturata che identifica la comunicazione. Quando perviene la comunicazione dal client$_{|g|}$ amministratore, il metodo crea e ritorna un nuovo oggetto `WSAdmin.WebSocket` rimanendo accessibile per nuove richieste di connessione.

4.2.2.4 WSAdmin.WebSocket

Funzione:
La classe ha il compito di gestire le comunicazioni in entrata ed uscita tra i client$_{|g|}$ di tipo amministratore ed il server$_{|g|}$. Quando riceve un nuovo messaggio da parte del client$_{|g|}$ amministratore, la classe inoltra la stringa alle classi interpreti e queste ultime, in casi specifici, ritornano dei messaggi da inviare al client$_{|g|}$ amministratore.

Attributi:
- `WsOutbound outbound`: riferimento al client$_{|g|}$ amministratore destinatario del messaggio da inviare.

- `IManageWSA adminManage`: riferimento all'oggetto `mytalk.server.presenter.administrator.logicAdmin.IManageWSA` per l'interpretazione dei messaggi in ingresso e creazione dei messaggi d'invio.

Metodi:
`# WebSocket();`
Costruttore: crea il riferimento all'attributo `adminManage` verso l'oggetto `mytalk.server.presenter.administrator.logicAdmin.ManageWSA`.

`- void sendMessage(WsOutbound o, String m);`
Si occupa dell'invio dei messaggi all'utente amministratore. Il parametro `o` è un riferimento all'utente amministratore destinatario. Se il valore è `null`, l'utente al quale viene inviato il messaggio è lo stesso che ha instaurato la connessione specificato dall'attributo `outbound`.
Il parametro `m` è un messaggio da inviare all'utente amministratore.

`+ void onOpen(WsOutbound outbound);`
Gestisce la richiesta di aprire una nuova connessione da parte dell'utente amministratore. Il parametro `outbound` è il riferimento all'utente amministratore che ha effettuato la connessione.

`+ void onTextMessage(CharBuffer buffer) throws IOException;`
Gestisce i messaggi inviati dal client$_{|g|}$ amministratore restituendo un opportuno messaggio quando previsto. Il parametro `buffer` riferisce il contenuto testuale del messaggio. Il metodo inoltra il messaggio all'oggetto `mytalk.server.presenter.administrator.logicAdmin.ManageWSA`, riferito da `adminManage`, il quale elabora, verifica e ritorna le informazioni richieste dal messaggio.

4.2.2.5 ManageWSA

Funzione:
La classe ha la funzione di gestire le stringhe di messaggi ricevute in ingresso.

Ciascun messaggio deve avere una struttura basata su sintassi XML$_{|g|}$ prefissata così da poterne creare una struttura dati ordinata dalla quale estrapolare in modo agevole le informazioni. Le informazioni ottenute servono come parametri vincolanti per ottenere informazioni dal DAO per poi essere organizzate in messaggi, sempre in sintassi XML$_{|g|}$, da inviare al client$_{|g|}$ amministratore.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.presenter.administrator.logicAdmin.IManageWSA`.

Usa le classi:

`mytalk.server.presenter.XMLField`;

`mytalk.server.model.dao.DataAccessObject`;

`mytalk.server.model.dao.ObjectTransfert`.

tramite l'interfaccia:

`mytalk.server.model.dao.IDataAccessObject`;

`mytalk.server.model.dao.IObjectTransfert`.

Attributi:
- `static IDataAccessObject dao`: riferimento all'oggetto `mytalk.server.model.dao.DataAccessObject` che gestisce le comunicazioni con la base dati.

- `String operation`: riferimento al valore dell'ultima operazione effettuata.

Metodi:
`+ String getOperation();`
Metodo getter, ritorna il valore contenuto nell'attributo `operation`, ovvero, l'ultima operazione gestita dall'oggetto.

`+ String request(String xml);`
Elabora la stringa XML$_{|g|}$ fornita in ingresso. L'elaborazione della stringa XML$_{|g|}$ richiede la creazione di una struttura dati ordinata che ne consenta un'agile estrazione delle informazioni; tali informazioni servono poi per scegliere il metodo interno relativo all'operazione richiesta. Come risultato, il metodo ritorna una nuova stringa in formato XML$_{|g|}$ contente le informazioni richieste.

`- String operationLog(Element e);`
Gestisce i dati di login dell'utente amministratore. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, aggiornandone i valori, il metodo compila il messaggio XML$_{|g|}$ con le informazioni ricavate.

- `String operationListUser(Element e);`
Gestisce la richiesta di ottenimento della lista di tutti gli utenti registrati. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

- `String operationExtractStat(Element e);`
Gestisce i dati di ricerca delle statistiche di comunicazione. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

- `String operationListUser(Element e);`
Gestisce la richiesta di ottenimento della lista di tutti gli utenti registrati. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

4.2.3 Package mytalk.server.presenter.user.logicUser

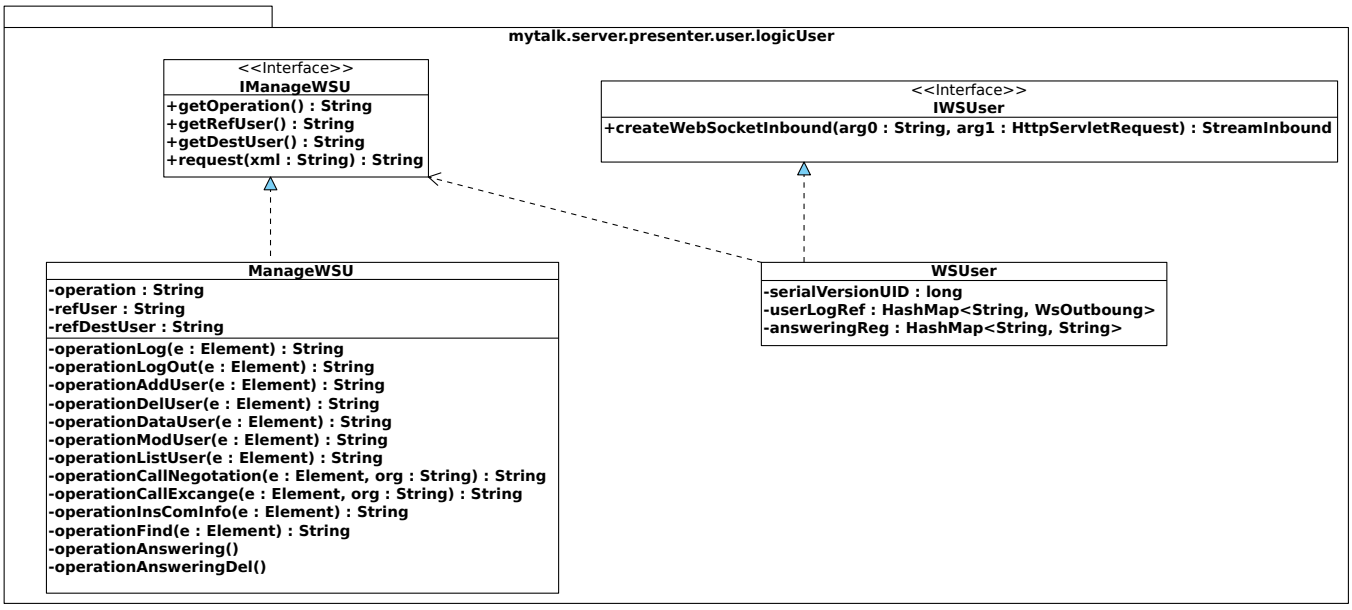

Figura 16: Diagramma delle classi del package `mytalk.server.presenter.user.logicUser`; dettaglio delle classi `IWSUser`, `IManageWSU`, `WSUser` e `ManageWSU`.

4.2.3.1 IWSUser

Funzione:
Interfaccia che gestisce le connessioni da parte dei client$_{|g|}$ e mantiene attivo il riferimento per l'ingresso e l'uscita di messaggi da/per il client$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.presenter.user.logicUser.WSUser`.

Metodi:
`+ createWebSocketInbound(String arg0, HttpServletRequest arg1);`
Ad ogni sua invocazione, crea l'oggetto `WSAdmin.WebSocket`.

4.2.3.2 IManageWSU

Funzione:
Interfaccia che gestisce i messaggi inviati dall'utente elaborandoli ed estrapolandone le informazioni di ricerca per poi restituirne un messaggio, correttamente composto, all'utente.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.presenter.user.logicUser.ManageWSU`.

Metodi:
`+ String getOperation();`
Ritorna il nome dell'ultima elaborazione effettuata dall'oggetto.

`+ String getRefUser();`
Ritorna il valore dell'e-mail dell'utente mittente del corrente messaggio.

`+ String getDestUser();` Ritorna il valore dell'e-mail dell'utente destinatario del corrente messaggio.

`+ String request(String xml);` Elabora la stringa in ingresso selezionando l'operazione associata da eseguire (quella richiesta dal messaggio) restituendo il messaggio risultante.

4.2.3.3 WSUser

Funzione:
Serve da riferimento per le comunicazioni client-server$_{|g|}$. La classe deve inoltre creare l'oggetto `logicUser.WebSocket` per l'invio la e ricezione delle

comunicazioni e mantenere una lista dei client$_{|g|}$ attualmente connessi al server$_{|g|}$.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.presenter.administrator.logicAdmin.IWSUser`.

Usa le classi:

`mytalk.server.presenter.user.logicUser.ManageWSU`;

tramite l'interfaccia:

`mytalk.server.presenter.user.logicUser.IManageWSU`.

Crea la classe:

`mytalk.server.presenter.user.logicUser.WSUser.WebSocket`.

Attributi:
- `static final long serialVersionUID`: valore per la classe seriale.

- `static HashMap<String, WsOutbound> userLogRef`: mappa degli utenti attualmente connessi attraverso la relazione e-mail - canale di riferimento.

- `static HashMap<String, String> answeringReg`: mappa degli utenti che hanno uno o più messaggi testuali pendenti da inviare.

Metodi:
`+ StreamInbound createWebSocketInbound(String arg0, HttpServletRequest arg1);`
Gestisce le richieste di connessione pervenute dai client$_{|g|}$. Il parametro `arg0` serve a specificare un sub-protocollo di comunicazione tra client$_{|g|}$ e server$_{|g|}$. Se non specificato, il suo valore è `null`. Il parametro `arg1` serve a riferire la richiesta HTTP$_{|g|}$ catturata che identifica la comunicazione. Quando perviene la comunicazione dal client$_{|g|}$, il metodo crea e ritorna un nuovo oggetto `WSUser.WebSocket` rimanendo accessibile per nuove richieste di connessione.

4.2.3.4 WSUser.WebSocket

Funzione:
La classe ha il compito di gestire le comunicazioni in entrata ed uscita tra i client$_{|g|}$ ed il server$_{|g|}$. Quando riceve un nuovo messaggio da parte del client$_{|g|}$, la classe inoltra la stringa alle classi interpreti. Queste ultime, in casi specifici, ritornano dei messaggi da inviare al client$_{|g|}$.

Attributi:
- `WsOutbound outbound`: riferimento al client$_{|g|}$ destinatario del messaggio da inviare.

- `String ip`: valore dell'IP$_{|g|}$ del client$_{|g|}$ che ha effettuato la connessione.

- `IManageWSU userManage`: riferimento all'oggetto `mytalk.server.presenter.user.logicUser.IManageWSU` per l'interpretazione dei messaggi in ingresso e creazione dei messaggi d'invio.

Metodi:
`# WebSocket(String ipAddress);`
Costruttore: aggiorna il valore dell'attributo `ip` e crea il riferimento all'attributo userManage verso l'oggetto `mytalk.server.presenter.user.logicUser.ManageWSU`.

`- void sendMessage(WsOutbound o, String m);`
Si occupa dell'invio dei messaggi all'utente designato. Il parametro `o` identifica un riferimento all'utente destinatario. Se il valore è `null`, l'utente al quale viene inviato il messaggio è lo stesso che ha instaurato la connessione. Il parametro `m` rappresenta il messaggio da inviare all'utente.

`+ void onOpen(WsOutbound outbound);`
Gestisce la richiesta di aprire una nuova connessione da parte dell'utente. Il parametro `outbound` è il riferimento all'utente. Il metodo si occupa anche di inviare all'utente richiedente un messaggio contente il suo indirizzo IP$_{|g|}$.

`+ void onTextMessage(CharBuffer buffer) throws IOException;`
Gestisce i messaggi inviati dal client$_{|g|}$ restituendo un opportuno messaggio quando previsto. Il parametro `buffer` riferisce il contenuto testuale del messaggio. Il metodo inoltra il messaggio all'oggetto `mytalk.server.presenter.user.logicUser.ManageWSU`, riferito da `userManage`, il quale elabora, verifica e ritorna le informazioni richieste dal messaggio. Gestisce inoltre quale operazione eseguire ad ogni richiesta del client$_{|g|}$ e gli ritorna un messaggio quando previsto.

`# void updateListUser(String u);`
Gestisce l'aggiornamento agli utenti dei nuovi utenti registrati. Il parametro `u` identifica l'email dell'ultimo utente autenticato.

`# String answering(String m);`
Gestisce i messaggi della segreteria. Il parametro `m` contiene il nuovo messaggio da inserire in segreteria. L'utente di riferimento viene ricavato dal messaggio stesso.

4.2.3.5 ManageWSU

Funzione:
La classe ha la funzione di gestire le stringhe di messaggi ricevute in ingresso. Ciascun messaggio deve avere una struttura basata su sintassi $XML_{|g|}$ prefissata così da poterne creare una struttura dati ordinata dalla quale estrapolare in modo agevole le informazioni. Le informazioni ottenute servono come parametri vincolanti per ottenere informazioni dal DAO per poi essere organizzate in messaggi, sempre in sintassi $XML_{|g|}$, da inviare al $client_{|g|}$.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.presenter.administrator.logicAdmin.IManageWSA`.

Usa le classi:

`mytalk.server.presenter.XMLField`;

`mytalk.server.model.dao.DataAccessObject`;

`mytalk.server.model.dao.ObjectTransfert`;

tramite l'interfaccia:

`mytalk.server.model.dao.IDataAccessObject`;

`mytalk.server.model.dao.IObjectTransfert`.

Attributi:
- `static IDataAccessObject dao`: riferimento all'oggetto `mytalk.server.model.dao.DataAccessObject` che gestisce le comunicazioni con la base dati.

- `String operation`: riferimento al valore dell'ultima operazione effettuata.

- `String refUser`: riferimento al valore e-mail dell'utente mittente (colui che genera l'oggetto).

- `String refDestUser`: riferimento al valore e-mail dell'utente destinatario.

Metodi:
`+ String getOperation();`
Metodo getter, ritorna il valore contenuto nell'attributo `operation`, ovvero l'ultima operazione gestita dall'oggetto.

`+ String getRefUser();`
Metodo getter, ritorna il valore contenuto nell'attributo `refUser`, ovvero, l'indirizzo e-mail dell'utente mittente dell'operazione.

`+ String getDestUser();`
Metodo getter, ritorna il valore contenuto nell'attributo `refDestUser`, ovvero, l'indirizzo e-mail dell'utente destinatario dell'operazione.

`+ String request(String xml);`
Elabora la stringa $XML_{|g|}$ fornita in ingresso. L'elaborazione della stringa $XML_{|g|}$ richiede la creazione di una struttura dati ordinata che ne consenta un'agile estrazione delle informazioni; tali informazioni servono poi per scegliere il metodo interno relativo all'operazione richiesta. Come risultato, il metodo ritorna una nuova stringa in formato $XML_{|g|}$ contente le informazioni richieste.

`- String operationLog(Element e);`
Gestisce i dati di login dell'utente. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, aggiornandone i valori, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

`- String operationLogOut(Element e);`
Gestisce i dati di logout dell'utente. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Il metodo effettua l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject` per aggiornare i valori.

`- String operationAddUser(Element e);`
Gestisce i dati per l'aggiunta di un nuovo utente. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare ed inserire. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, aggiornandone i valori, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

`- String operationDelUser(Element e);`
Gestisce i dati per l'eliminazione di un utente registrato. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare ed eliminare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, aggiornandone i valori, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

`- String operationDataUser(Element e);`
Gestisce i dati per ottenere le informazioni relative all'utente selezionato. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

`- String operationModUser(Element e);`
Gestisce i dati per modificare le informazioni relative all'utente selezionato.

Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare ed inserire. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, aggiornandone i valori, il metodo compila il messaggio $XML_{|g|}$ con l'esito dell'operazione.

- `String operationListUser(Element e);`
Gestisce la richiesta di ottenimento della lista di tutti gli utenti registrati. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Una volta fatta l'interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con le informazioni ricavate.

- `String operationCallNegotation(Element e, String org);`
Gestisce la richiesta di negoziazione di comunicazione tra due utenti. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Il metodo deve poter verificare se l'utente destinatario è online tramite un' interrogazione all'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`. In caso affermativo, la stringa originaria, cioè il parametro `org`, non viene modificata, altrimenti viene segnalato che l'utente è offline.

- `String operationCallExcange(Element e, String org);`
Gestisce la richiesta di scambio delle informazioni di comunicazione tra due utenti. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare. Il metodo aggiorna l'attributo `refDestUser` e restituisce la stringa $XML_{|g|}$ originaria.

- `String operationInsComInfo(Element e);`
Gestisce i dati statistici delle comunicazioni. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare ed inserire. Una volta inseriti i valori indicati tramite l'oggetto di tipo `mytalk.server.model.dao.DataAccessObject`, il metodo compila il messaggio $XML_{|g|}$ con l'esito dell'operazione.

- `String operationFind(Element e);`
Gestisce i dati di ricerca per le statistiche delle comunicazioni. Il parametro `e` riferisce la struttura dati che gestisce le informazioni da verificare ed inserire. Una volta verificate le informazioni inserite, il metodo interroga l'oggetto di tipo `mytalk.server.model.dao.DataAccessObject` per ottenere il risultato di ricerca che viene composto in una stringa $XML_{|g|}$.

- `String operationAnswering(Element e, String org);`
Gestisce i dati del nuovo messaggio di segreteria.
Il parametro `e` riferisce la struttura dati che gestisce le informazioni da estrapolare. Una volta estrapolati i riferimenti email degli utenti mittenti e destinatari, il metodo ritorna il messaggio originario (definito dal parametro `org`).

- `String operationAnsweringDel(Element e);`
Gestisce i dati del messaggio di segreteriada eliminare.
Estrapola il riferimento email dell'utente destinatario, il metodo ritorna il messaggio originario (definito dal parametro `org`).

5 Specifica della componente Model

La componente Model gestisce tutte le informazioni permanenti dell'applicazione, questa si suddivide in due categorie Client$_{|g|}$ (vedi 5.1) e Server$_{|g|}$ (vedi 5.2).

5.1 Client

La sotto-componente Client$_{|g|}$ si occupa della gestione dei cookie$_{|g|}$ di sessione, ovvero della loro creazione, del loro recupero e della loro distruzione. È formata dalle classi:

`mytalk.client.model.localDataUser.ManageCookies` (sez. 5.1.1.1)

`mytalk.client.administrator.localDataUser.ManageCookies` (sez. 5.1.2.1)

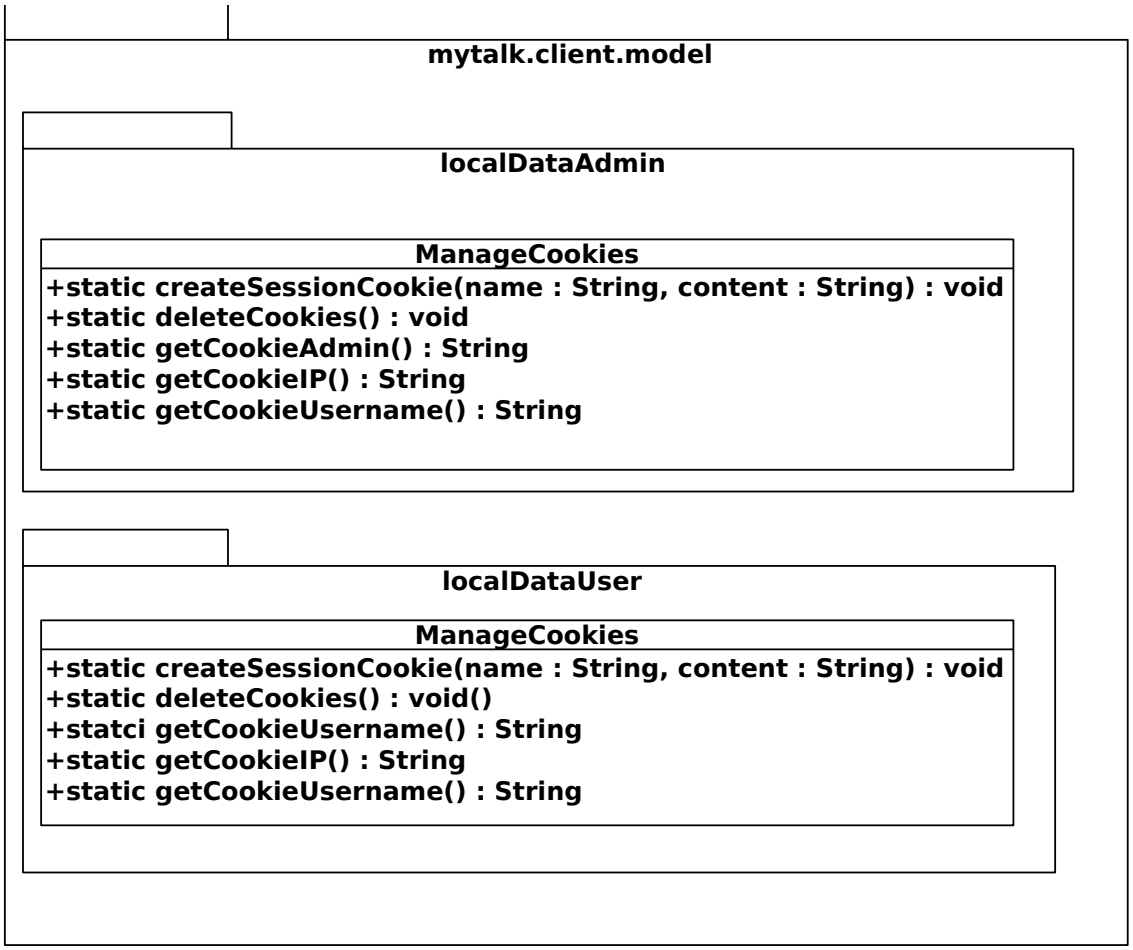

Figura 17: Diagramma delle classi del package `mytalk.client.model`; dettaglio delle classi `localDataAdmin.ManageCookies` e `localDataUser.ManageCookies`.

5.1.1 Package mytalk.client.model.localDataUser

5.1.1.1 ManageCookies

Funzione:
Classe astratta che contiene la gestione dei cookie$_{|g|}$.

Relazioni con altre componenti:
I suoi metodi vengono usati da:

`mytalk.client.presenter.user.logicUser.LogUserLogic`;

`mytalk.client.presenter.user.logicUser.RegisterLogic`;

`mytalk.client.presenter.user.logicUser.DataUserLogic`;

`mytalk.client.presenter.user.logicUser.UpdateViewLogic`;

`mytalk.client.presenter.administrator.logicAdmin.LogAdminLogic`;

`mytalk.client.presenter.administrator.serverComAdmin.WebSocketAdmin`;

`mytalk.client.presenter.user.serverComUser.WebSocketUser`.

Attributi:
Nessuno.

Metodi:
`+ static void createSessionCookie(String name, String content);`
Crea il cookie$_{|g|}$ di sessione associato all'utente. Viene richiamata due volte, una per creare un cookie$_{|g|}$ con lo username e una per creare un cookie$_{|g|}$ con l'indirizzo IP$_{|g|}$ dell'utente.

`+ static String getCookieUsername();`
Ritorna il nome utente associato all'utente autenticato.

`+ static String getCookieIP();`
Ritorna l'indirizzo IP dell'utente attualmente autenticato.

`+ static void deleteCookies();`
Azzera e successivamente rimuove all'aggiornamento della pagina tutti i cookie$_{|g|}$ associati all'utente.

5.1.2 Package mytalk.client.administrator.localDataUser

5.1.2.1 ManageCookies

Funzione:
Classe astratta che contiene la gestione dei cookie$_{|g|}$.

Relazioni con altre componenti:
I suoi metodi vengono usati da:

`mytalk.client.presenter.administrator.logicAdmin.StatisticLogic`;

`mytalk.client.presenter.administrator.logicAdmin.UpdateViewLogic`;

Attributi:
Nessuno.

Metodi:
`+ static void createSessionCookie(String name, String content);`
Crea il cookie$_{|g|}$ di sessione associato all'utente. Viene richiamata due volte, una per creare un cookie$_{|g|}$ con lo username e una per creare un cookie$_{|g|}$ con l'indirizzo IP$_{|g|}$ dell'utente.

`+ static String getCookieUsername();`
Ritorna il nome utente associato all'utente autenticato.

`+ static String getCookieIP();`
Ritorna l'indirizzo IP dell'utente attualmente autenticato.

`+ static void deleteCookies();`
Azzera e successivamente rimuove all'aggiornamento della pagina tutti i cookie$_{|g|}$ associati all'utente.

5.2 Server

La sotto-componente Server$_{|g|}$ permette la gestione e la fornitura delle informazioni ricevute da/per altre componenti in una struttura DBMS$_{|g|}$ ordinata. È formata dalle classi:

`mytalk.server.model.dao.IObjectTransfer` (sez. 5.2.1.1)

`mytalk.server.model.dao.IDataAccessObject` (sez. 5.2.1.2)

`mytalk.server.model.dao.ObjectTransfer` (sez. 5.2.1.3)

`mytalk.server.model.dao.DataAccessObject` (sez. 5.2.1.4)

`mytalk.server.model.dao.DataAccessObject.User` (sez. 5.2.1.5)

`mytalk.server.model.dao.DataAccessObject.Comm` (sez. 5.2.1.6)

`mytalk.server.model.dao.DataAccessObject.DBDataAccess` (sez. 5.2.1.7)

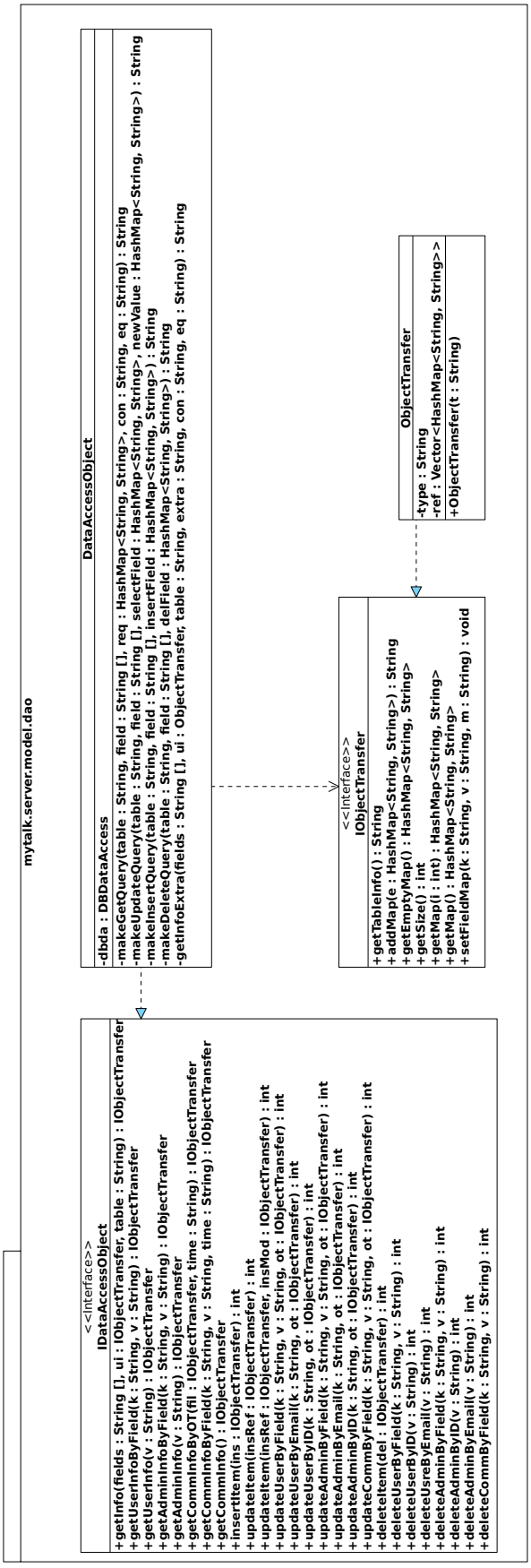

Figura 18: Diagramma delle classi del package `mytalk.server.model.dao`; dettaglio delle classi `IDataAccessObject`, `IObjectTransfert`, `DataAccessObject` e `ObjectTransfert`

5.2.1 Package mytalk.server.model.dao

5.2.1.1 IObjectTransfer

Funzione:
Fornisce una struttura dati univoca per la gestione dei valori ricavati e da inserire nel DBMS$_{|g|}$.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.model.dao.ObjectTransfer`.

L'interfaccia è utilizzata da:

`mytalk.server.model.dao.DataAccessObject`;

`mytalk.server.presenter.XMLField`;

`mytalk.server.presenter.user.logicUser.ManageWSU`;

`mytalk.server.presenter.admin.logicAdmin.ManageWSA`.

Metodi:
`+ String getTableInfo();`
Restituisce una stringa che specifica il tipo di dati che l'oggetto contiene.

`+ void addMap(HashMap<String, String> e);`
Aggiunge all'oggetto una nuova mappa Hash con informazioni specifiche.

`+ HashMap<String, String> getEmptyMap();`
Restituisce una mappa Hash vuota dello stesso tipo dell'oggetto d'invocazione.

`+ int getSize();`
Restituisce il numero di mappe Hash contenute nell'oggetto.

`+ HashMap<String, String> getMap(int i);`
Restituisce il riferimento all' i-esima mappa Hash contenuta nell'oggetto.

`+ HashMap<String, String> getMap();`
Restituisce il riferimento alla prima mappa Hash contenuta nell'oggetto.

`+ void setFieldMap(String k, String v, int m);`
Modifica un particolare valore della m-esima mappa Hash attraverso il riferimento k ed il valore associato v.

5.2.1.2 IDataAccessObject

Funzione:
Esegue le operazioni richieste verso il $DBMS_{|g|}$ scelto ritornandone le informazioni ricavate.

Relazioni con altre componenti:
L'interfaccia è implementata da:

`mytalk.server.model.dao.DataAccessObject`.

L'interfaccia è utilizzata da:

`mytalk.server.presenter.user.logicUser.ManageWSU`;

`mytalk.server.presenter.admin.logicAdmin.ManageWSA`.

Metodi:
`+ ObjectTransfer getInfo(String[] fields, ObjectTransfer ui, String table);`
Restituisce le informazioni richieste mediante i vincoli indicati dai parametri d'ingresso.

`+ ObjectTransfer getUserInfoByField(String k, String v);`
Restituisce le informazioni richieste relative all'utente mediante un unico vincolo relazionale chiave – valore.

`+ ObjectTransfer getUserInfo(String v);`
Restituisce le informazioni richieste relative all'utente mediante specifica del valore e-mail.

`+ ObjectTransfer getAdminInfoByField(String k, String v);`
Restituisce le informazioni richieste relative all'amministratore mediante un unico vincolo relazionale chiave – valore.

`+ ObjectTransfer getAdminInfo(String v);`
Restituisce le informazioni richieste relative all'amministratore mediante specifica del valore e-mail.

`+ ObjectTransfer getCommInfoByOT(ObjectTransfer fil, String time);`
Restituisce le informazioni relative alle comunicazioni filtrate dalle informazioni passate ed il tempo di interesse.

`+ ObjectTransfer getCommInfoByField(String k, String v);`
Restituisce le informazioni relative alle comunicazioni mediante un unico vincolo relazionale chiave – valore.

`+ ObjectTransfer getCommInfo();`
Restituisce le informazioni relative a tutte le comunicazioni presenti nel $DBMS_{|g|}$.

`+ int insertItem(ObjectTransfer ins);`
Inserisce nel $DBMS_{|g|}$ le informazioni presenti nell'oggetto passato.

`+ int updateItem(ObjectTransfer insRef, ObjectTransfer insMod);`
Modifica le informazioni contenute nell'oggetto riferito da parametro `insRef` presenti nel $DBMS_{|g|}$ aggiornandole con quelle indicate dal parametro `insMod`.

`+ int updateUserByField(String k, String v, ObjectTransfer ot);`
Modifica le informazioni dell'utente presenti nel $DBMS_{|g|}$, riferite dalla coppia chiave – valore passata in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateUserByEmail(String k, ObjectTransfer ot);`
Modifica le informazioni dell'utente presenti nel $DBMS_{|g|}$, riferite dal valore e-mail dato in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateUserByID(String k, ObjectTransfer ot);`
Modifica le informazioni dell'utente presenti nel $DBMS_{|g|}$, riferite dal valore ID dell'utente dato in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateAdminByField(String k, String v, ObjectTransfer ot);`
Modifica le informazioni dell'amministratore presenti nel $DBMS_{|g|}$, riferite dalla coppia chiave – valore passata in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateAdminByEmail(String k, ObjectTransfer ot);`
Modifica le informazioni dell'amministratore presenti nel $DBMS_{|g|}$, riferite dal valore e-mail dato in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateAdminByID(String k, ObjectTransfer ot);`
Modifica le informazioni dell'amministratore presenti nel $DBMS_{|g|}$, riferite dal valore ID dell'utente dato in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int updateCommByField(String k, String v, ObjectTransfer ot);`
Modifica le informazioni della comunicazione presenti nel $DBMS_{|g|}$, riferite dalla coppia chiave – valore passata in ingresso, aggiornandole con quelle indicate dal parametro `ot`.

`+ int deleteItem(ObjectTransfer del);`
Elimina dal DBMS$_{|g|}$ le informazioni riferite dal riferimento passato.

`+ int deleteUserByField(String k, String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'utente riferite dalla coppia chiave – valore passata in ingresso.

`+ int deleteUserByID(String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'utente riferite dal valore dell'ID dato in ingresso.

`+ int deleteUserByEmail(String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'utente riferite dal valore dell'e-mail dato in ingresso.

`+ int deleteAdminByField(String k, String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'amministratore riferite dalla coppia chiave – valore passata in ingresso.

`+ int deleteAdminByID(String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'amministratore riferite dal valore dell'ID dato in ingresso.

`+ int deleteAdminByEmail(String v);`
Elimina dal DBMS$_{|g|}$ le informazioni dell'amministratore riferite dal valore dell'e-mail dato in ingresso.

`+ int deleteCommByField(String k, String v);`
Elimina dal DBMS$_{|g|}$ le informazioni della comunicazione riferite dalla coppia chiave – valore passata in ingresso.

5.2.1.3 ObjectTransfer

Funzione:
La classe ha la funzione di creare un riferimento univoco ai dati manipolati dal DAO. La sua composizione è incentrata sulla gestione di mappe Hash le quali assumono una tipologia ben definita, a seconda dei parametri di riferimento.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.model.dao.IObjectTransfer`.

Usa le classi:

`mytalk.server.model.dao.DataAccessObject`

tramite l'interfaccia:

`mytalk.server.model.dao.IDataAccessObject`.

Attributi:
- `Vector<HashMap<String, String>> ref`: vettore delle mappe Hash manipolate dai metodi dell'oggetto.

- `String type`;: identifica il tipo di dati contenuti nell'oggetto.

Metodi:
`+ ObjectTransfer(String t);`
Costruttore: inizializza l'oggetto `ObjectTransfer` del tipo specificato dalla stringa in ingresso. La tipologia resta immutabile per tutta la vita dell'oggetto.

`+ ObjectTransfer(String t, HashMap<String, String> e);`
Costruttore: inizializza l'oggetto `ObjectTransfer` del tipo specificato ed inserendo già una mappa Hash dello stesso tipo al suo interno. La tipologia resta immutabile per tutta la vita dell'oggetto.

`+ String getTableInfo();`
Restituisce una stringa che specifica il tipo di dati che l'oggetto contiene. Questo metodo è puramente informativo.

`+ void addMap(HashMap<String, String> e);`
Consente l'aggiunta di una nuova mappa Hash nell'oggetto a patto che essa sia dello stesso tipo dell'oggetto.

`+ HashMap<String, String> getEmptyMap();`
Consente di ottenere una mappa Hash vuota dello stesso tipo dell'oggetto `ObjectTransfer`.

`+ int getSize();`
Restituisce il numero di mappe Hash presenti nell'oggetto. Si limita a restituire il numero di elementi inseriti nel vettore che gestisce le mappe Hash.

`+ HashMap<String, String> getMap(int i);`
Restituisce la mappa Hash indicata dall'indice intero `i`. Se viene selezionato un indice negativo o maggiore del numero di mappe contenute nell'oggetto, il valore di ritorno è `null`.

`+ HashMap<String, String> getMap();`
Scorciatoia per ottenere la prima mappa Hash contenuta nell'oggetto.

`+ void setFieldMap(String k, String v, int m);`
Modifica un particolare riferimento nella mappa selezionata. L'indice `m`, che deve essere valido ($0 \leqslant$ `m` $<$`this.getSize()`) seleziona la mappa alla quale apportare la modifica. Il parametro `k` specifica la chiave di riferimento, mentre il parametro `v` il valore ad essa associata. Viene lasciata libertà all'utente, in quanto non viene controllato che la chiave di riferimento sia inerente al tipo della mappa Hash.

5.2.1.4 DataAccessObject

Funzione:
La classe ha la funzione di interagire con il DBMS$_{|g|}$ selezionato. Questa classe, inoltre raggruppa i vari riferimenti ai campi informativi, usati per gestire le informazioni ricavate dal DBMS$_{|g|}$, nelle due classi interne `DataAccessObject.User` e `DataAccessObject.Comm`. La gestione della connessione con il DBMS$_{|g|}$ avviene mediante la classe interna `DataAccessObject.DBDataAccess`. Tutti i metodi sviluppati sono concepiti per consentire più modi per interagire con il DBMS$_{|g|}$ in modo da semplificarne l'utilizzo.

Relazioni con altre componenti:
Implementa l'interfaccia:

`mytalk.server.model.dao.IDataAccessObject`.

Usa le classi:

`mytalk.server.model.dao.ObjectTransfer`

e crea le classi:

`mytalk.server.model.dao.DataAccessObject.User`;

`mytalk.server.model.dao.DataAccessObject.Comm`;

`mytalk.server.model.dao.DataAccessObject.DBDataAccess`.

Attributi:
`- DBDataAccess dbda`: riferimento alla connessione al DBMS$_{|g|}$ per inoltrare le query$_{|g|}$ prodotte dai vari metodi.

Metodi:
`+ DataAccessObject();`
Costruttore: crea un oggetto di tipo `DBDataAccess` per la connessione al DBMS$_{|g|}$.

`- String makeGetQuery(String table, String[] field,`
`HashMap<String, String> req, String con);`
Crea e ritorna la stringa query$_{|g|}$ da inoltrare al DBMS$_{|g|}$ per ottenere le

informazioni richieste. La stringa viene personalizzata dai parametri che specificano:

- `table`: la tabella dove reperire le informazioni;
- `field`: i campi informativi da reperire;
- `req`: elenco delle coppie chiave – valore da usare come riferimento (se `null`, vengono restituite tutte le informazioni disponibili).
- `con`: specifica il connettivo di clausola per filtrare i risultati (`AND` di default).

`- String makeUpdateQuery(String table, String[] field, HashMap<String, String> selectField, HashMap<String, String> newValue);`
Crea e ritorna la stringa $query_{|g|}$ da inoltrare al $DBMS_{|g|}$ per aggiornare i valori selezionati. La stringa viene personalizzata dai parametri d'ingresso che specificano:

- `table`: la tabella dove aggiornare le informazioni;
- `field`: i campi informativi da aggiornare;
- `selectField`: elenco delle coppie chiave – valore da usare come riferimento;
- `newValue`: elenco delle coppie chiave – valore da inserire come valori da aggiornare.

`- String makeInsertQuery(String table, String[] field, HashMap<String, String> insertField);`
Crea e ritorna la stringa $query_{|g|}$ da inoltrare al $DBMS_{|g|}$ per inserire le nuove informazioni inoltrate. La stringa viene personalizzata dai parametri d'ingresso che specificano:

- `table`: la tabella dove inserire le informazioni;
- `field`: i campi informativi da inserire;
- `insertField`: elenco delle coppie chiave – valore da inserire come nuovi valori.

`- String makeDeleteQuery(String table, String[] field, HashMap<String, String> delField);`
Crea e ritorna la stringa $query_{|g|}$ da inoltrare al $DBMS_{|g|}$ per eliminare le informazioni inoltrate. La stringa viene personalizzata dai parametri d'ingresso che specificano:

- `table`: la tabella dove eliminare le informazioni;
- `field`: i campi informativi da eliminare;
- `delField`: elenco delle coppie chiave – valore da eliminare.

`- ObjectTransfer getInfoExtra(String[] fields, ObjectTransfer ui, String table, String extra, String con, String eq);`
Restituisce, mediante oggetto `mytalk.server.presenter.user.logicUser.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `fields`: elenco chiavi per le informazioni da ricavare;
- `ui`: valori di riferimento per filtrare le informazioni cercate;
- `table`: tabella dove ricercare le informazioni;
- `extra`: stringa aggiuntiva alla $query_{|g|}$ che contiene parametri di filtraggio specifici;
- `con`: stringa che specifica il connettivo di collegamento delle clausole di filtro per il parametro `WHERE` della $query_{|g|}$;
- `eq`: stringa che specifica il connettivo di raffronto della clausola di filtro per il parametro `WHERE` della $query_{|g|}$.

Nel metodo viene fatta richiesta per la creazione della $query_{|g|}$ d'informazione (attraverso il metodo `makeGetQuery`) e per l'esecuzione della $query_{|g|}$ (oggetto DataAccessObject.DBDataAccess).

`+ ObjectTransfer getInfo(String[] fields, ObjectTransfer ui, String table);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$. Il metodo è un'alternativa al metodo `getInfoExtra`.

`+ ObjectTransfer getUserInfoByField(String k, String v);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative agli utenti. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli utenti.

Questo metodo fa uso del più generico metodo `getInfo` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ ObjectTransfer getUserInfo(String v);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative agli utenti. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `v`: valore e-mail di riferimento per selezionare l'utente.

Questo metodo fa uso del più generico metodo `getInfo` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ ObjectTransfer getAdminInfoByField(String k, String v);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative agli amministratori. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli amministratori.

Questo metodo fa uso del più generico metodo getInfo delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ ObjectTransfer getAdminInfo(String v);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative agli amministratori. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `v`: valore e-mail di riferimento per selezionare l' amministratore.

Questo metodo fa uso del più generico metodo `getInfo` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ ObjectTransfer getCommInfoByOT(ObjectTransfer fil, String time);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative alle statistiche di comunicazione. Il metodo estrapola i vincoli d'informazione dall'oggetto in ingresso:

- `fil`: Riferimento all'oggetto `mytalk.server.model.dao.ObjectTranfer` contenente i parametri di filtro;
- `time`: valore dell'arco temporale d'interessamento delle statistiche d'informazione. Se il suo valore è `null`, ritorna le informazioni disponibili seguendo i vincoli indicati.

`+ ObjectTransfer getCommInfoByField(String k, String v, String time);`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, le informazioni richieste presenti nel $DBMS_{|g|}$ relative alle statistiche di comunicazione. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni delle comunicazioni;
- `time`: valore dell'arco temporale d'interessamento delle statistiche d'informazione. Se il suo valore è null, ritorna le informazioni disponibili seguendo i vincoli indicati.

`+ ObjectTransfer getCommInfo();`
Restituisce, mediante oggetto `mytalk.server.model.dao.ObjectTranfer`, tutte le informazioni presenti nel DBMS$_{|g|}$ relative alle statistiche di comunicazione. Questo metodo fa uso del più generico metodo `getCommInfoByField` delegandone le chiamate ai metodi interni per la query$_{|g|}$ e la sua esecuzione al DBMS$_{|g|}$.

`+ int insertItem(ObjectTransfer ins);`
Inserisce nel DBMS$_{|g|}$ le informazioni estrapolate dall'oggetto `mytalk.server.model.dao.ObjectTranfer ins` passato in ingresso nella tabella specificata dall'oggetto stesso. Il numero di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel DBMS$_{|g|}$. Nel metodo viene fatta richiesta per la creazione della query$_{|g|}$ di inserimento (attraverso il metodo `makeInsertQuery`) e di eseguire la query$_{|g|}$ (oggetto `DataAccessObject.DBDataAccess`).

`+ int updateItem(ObjectTransfer insRef, ObjectTransfer insMod);`
Aggiorna le informazioni selezionate presenti nel DBMS$_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati in ingresso:

- `insRef`: oggetto da cui estrapolare le informazioni di vincolo per l'aggiornamento della tabella soggetta (tipologia estrapolata dall'oggetto stesso);
- `insMod`: oggetto contenente le modifiche da apportare al DBMS$_{|g|}$.

La tipologia di tabella dove effettuare la modifica deve essere la medesima per entrambi gli oggetti. Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel DBMS$_{|g|}$. Nel metodo viene fatta richiesta per la creazione della query$_{|g|}$ di aggiornamento (attraverso il metodo `makeUpdateQuery`) e di eseguire la query$_{|g|}$ (oggetto `DataAccessObject.DBDataAccess`).

`+ int updateUserByField(String k, String v, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli utenti presenti nel DBMS$_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli utenti;
- `ot`: oggetto contenente le modifiche da apportare al DBMS$_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel DBMS$_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la query$_{|g|}$ e la sua esecuzione al DBMS$_{|g|}$.

`+ int updateUserByEmail(String k, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli utenti presenti nel DBMS$_{|g|}$ con

dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: valore di riferimento dell'e-mail per filtrare le informazioni degli utenti;
- `ot`: oggetto contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int updateUserByID(String k, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli utenti presenti nel $DBMS_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: valore di riferimento dell'ID per filtrare le informazioni degli utenti;
- `ot`: oggetto d'ingresso contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int updateAdminByField(String k, String v, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli amministratori presenti nel $DBMS_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli amministratori;
- `ot`: oggetto d'ingresso contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int updateAdminByEmail(String k, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli amministratori presenti nel $DBMS_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: valore di riferimento dell'e-mail per filtrare le informazioni degli amministratori;
- `ot`: oggetto d'ingresso contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int updateAdminByID(String k, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative agli amministratori presenti nel $DBMS_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: valore di riferimento dell'ID per filtrare le informazioni degli amministratori;
- `ot`: oggetto d'ingresso contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int updateCommByField(String k, String v, ObjectTransfer ot);`
Aggiorna le informazioni selezionate relative alle statistiche di comunicazione presenti nel $DBMS_{|g|}$ con dei nuovi valori passati. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni delle comunicazioni;
- `ot`: oggetto d'ingresso contenente le modifiche da apportare al $DBMS_{|g|}$.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `updateItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteItem(ObjectTransfer del);`
Elimina le informazioni selezionate presenti nel $DBMS_{|g|}$. Il metodo estrapola i vincoli di selezione dal parametro del quale ricava anche la tabella su cui eseguire l'operazione. Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Nel metodo viene fatta richiesta per la creazione della $query_{|g|}$ di eliminazione (attraverso il metodo `makeDeleteQuery`), la $query_{|g|}$ viene eseguita attraverso (oggetto `DataAccessObject.DBDataAccess`).

`+ int deleteUserByField(String k, String v);`
Elimina le informazioni relative agli utenti presenti nel $DBMS_{|g|}$. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli utenti.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteUserByID(String v);`
Elimina le informazioni relative agli utenti presenti nel $DBMS_{|g|}$ mediante parametro `v` che indica l'ID univoco. Il valore di ritorno si riferisce al numero

di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteUserByEmail(String v);`
Elimina le informazioni relative agli utenti presenti nel $DBMS_{|g|}$ mediante parametro `v` che indica l'indirizzo e-mail univoco. Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteAdminByField(String k, String v);`
Elimina le informazioni relative agli amministratori presenti nel $DBMS_{|g|}$. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni degli amministratori.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteAdminByID(String v);`
Elimina le informazioni relative agli amministratori presenti nel $DBMS_{|g|}$ mediante parametro `v` che indica l'ID univoco. Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteAdminByEmail(String v);`
Elimina le informazioni relative agli amministratori presenti nel $DBMS_{|g|}$ mediante parametro `v` che indica l'indirizzo e-mail univoco. Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

`+ int deleteCommByField(String k, String v);`
Elimina le informazioni relative alle statistiche di comunicazione presenti nel $DBMS_{|g|}$. Il metodo estrapola i vincoli d'informazione dai parametri passati:

- `k`: chiave di riferimento;
- `v`: valore di riferimento per filtrare le informazioni delle comunicazioni.

Il valore di ritorno si riferisce al numero di inserimenti avvenuti correttamente nel $DBMS_{|g|}$. Questo metodo fa uso del più generico metodo `deleteItem` delegandone le chiamate ai metodi interni per la $query_{|g|}$ e la sua esecuzione al $DBMS_{|g|}$.

5.2.1.5 DataAccessObject.User

Funzione:
La classe ha la funzione di racchiudere tutti i valori di riferimento relativi alle informazioni degli utenti e degli amministratori gestite nel DBMS$_{|g|}$ e nell'oggetto di scambio `mytalk.server.model.dao.ObjectTranfer`.

Relazioni con altre componenti:
La classe è implementata da:

- `mytalk.server.model.dao.DataAccessObject`;
- `mytalk.server.model.dao.ObjectTransfer`;
- `mytalk.server.presenter.XMLField`;
- `mytalk.server.presenter.user.logicUser.ManageWSU`;
- `mytalk.server.presenter.admin.logicAdmin.ManageWSA`.

Attributi:
`+ static final String user`: riferimento alla tabella degli utenti del DBMS$_{|g|}$.

`+ static final String admin`: riferimento alla tabella degli amministratori del DBMS$_{|g|}$.

`+ static final String fieldAID`: riferimento al valore ID della tabella degli amministratori.

`+ static final String fieldUID`: riferimento al valore ID della tabella degli utenti.

`+ static final String fieldPsw`: riferimento al valore password nelle tabelle degli utenti ed amministratori.

`+ static final String fieldName`: riferimento al valore nome nelle tabelle degli utenti ed amministratori.

`+ static final String fieldSName`: riferimento al valore cognome nelle tabelle degli utenti ed amministratori.

`+ static final String fieldMail`: riferimento al valore dell'indirizzo e-mail nelle tabelle utenti ed amministratori.

`+ static final String fieldDtReg`: riferimento al valore della data di registrazione nelle tabelle degli utenti ed amministratori.

`+ static final String fieldSociety`: riferimento al valore dell'azienda nelle tabelle degli utenti ed amministratori.

`+ static final String fieldTel`: riferimento al valore del numero telefonico nelle tabelle degli utenti ed amministratori.

`+ static final String fieldOnline`: riferimento al valore online nelle tabelle degli utenti.

`+ static final String fieldIP`: riferimento al valore IP nelle tabelle degli utenti.

`+ static final String[] fieldUser`: array che raccoglie tutte le informazioni relative agli utenti presenti nel $DBMS_{|g|}$.

`+ static final String[] fieldAdmin`: array che raccoglie tutte le informazioni relative agli amministratori presenti nel $DBMS_{|g|}$.

5.2.1.6 DataAccessObject.Comm

Funzione:
La classe ha la funzione di racchiudere tutti i valori relativi alle informazioni statistiche delle comunicazioni gestite nel $DBMS_{|g|}$ e nell'oggetto di scambio `mytalk.server.model.dao.ObjectTranfer`.

Relazioni con altre componenti:
La classe è implementata da:

- `mytalk.server.model.dao.DataAccessObject`;
- `mytalk.server.model.dao.ObjectTransfer`;
- `mytalk.server.presenter.XMLField`;
- `mytalk.server.presenter.user.logicUser.ManageWSU`;
- `mytalk.server.presenter.admin.logicAdmin.ManageWSA`.

Attributi:
`+ static final String comm`: riferimento alla tabella delle statistiche di comunicazione nel $DBMS_{|g|}$.

`+ static final String fieldCID`: riferimento al valore ID della tabella delle comunicazioni.

`+ static final String fieldUSender`: riferimento all'indirizzo e-mail dell'utente mittente nelle tabelle delle comunicazioni.

`+ static final String fieldUReceiver`: riferimento all'indirizzo e-mail dell'utente destinatario nelle tabelle delle comunicazioni.

`+ static final String fieldByteIn`: riferimento al numero di byte trasmessi nelle tabelle delle comunicazioni.

`+ static final String fieldPack`: riferimento al numero di pacchetti scambiati nelle tabelle delle comunicazioni.

`+ static final String fieldPackLost`: riferimento al numero di pacchetti persi nelle tabelle delle comunicazioni.

`+ static final String fieldStart`: riferimento alla data di inizio della chiamata nelle tabelle delle comunicazioni.

`+ static final String fieldEnd`: riferimento alla data di termine della chiamata nelle tabelle delle comunicazioni.

`+ static final String fieldGS`: riferimento al valore di gradimento dell'utente mittente nelle tabelle delle comunicazioni.

`+ static final String fieldGR`: riferimento al valore di gradimento dell'utente destinatario nelle tabelle delle comunicazioni.

`+ static final String[] fieldComm`: array che raccoglie tutti valori di riferimento per le informazioni relative alla comunicazione presenti nel $DBMS_{|g|}$.

5.2.1.7 DataAccessObject.DBDataAccess

Funzione:
La classe ha la funzione di gestire la connessione verso il $DBMS_{|g|}$ con i parametri indicati nel file `mytalk.server.model.dao.DBAccess.xml`.

Relazioni con altre componenti:
La classe è implementata da:

`mytalk.server.model.dao.DataAccessObject`.

Attributi:
`- String user`: riferimento al valore dell'utente abilitato alla connessione al $DBMS_{|g|}$.

- `String psw`: riferimento al valore della password utente abilitato alla connessione al DBMS$_{|g|}$.

- `String driver`: riferimento del driver usato dal programma per la connessione al DBMS$_{|g|}$.

- `String url`: riferimento URI usato dal programma per la connessione al DBMS$_{|g|}$.

- `Statement cmd`: stato della connessione al DBMS$_{|g|}$.

- `Connection con`: riferimento della connessione al DBMS$_{|g|}$.

Metodi:
`+ DBDataAccess();`
Costruttore: legge ed estrapola dal file `mytalk.server.model.dao.DBAccess.xml` le informazioni sufficienti per stabilire la connessione al DBMS$_{|g|}$.

`+ Statement open();`
Instaura la connessione al DBMS$_{|g|}$ secondo i parametri ricavati dal costruttore. Ritorna lo stato della connessione.

`+ void close(ResultSet res);`
Termina l'esecuzione della query$_{|g|}$ `res` e chiude la connessione al DBMS;

6 Digrammi di sequenza

6.1 Effettuazione chiamata

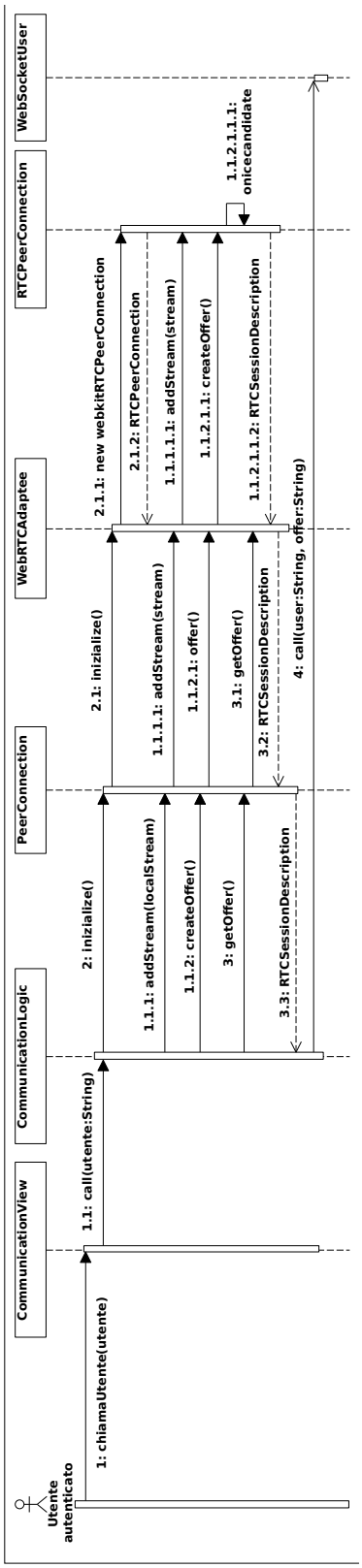

Figura 19: DS 1 - La procedura di effettuazione di una chiamata.

Precondizione: l'utente è autenticato e inserisce un nome utente esistente all'interno del sistema.

Descrizione: l'utente inserisce lo username dell'utente che vuole chiamare tramite un form o selezionandolo dalla lista degli utenti registrati al server$_{|g|}$. Vengono quindi eseguiti i seguenti passaggi:

1. viene invocato il metodo della classe `CommunicationLogic call(String utente)`;
2. il metodo invoca a sua volta il metodo `inizialize()` della classe `PeerConnection` che invoca il metodo omonimo nella classe wrapper `WebRTCAdaptee`;
3. il metodo `inizialize()` inizializza l'oggetto `pc` con l'oggetto di tipo `webkitPeerConnection` appena creato;
4. una volta ritornato dal metodo `inizialize()` la classe `CommunicationLogic` richiama il metodo `addStream(MediaStream stream)` dell'oggetto `connessione` di tipo `PeerConnection` con lo scopo di aggiungere lo stream video alla connessione, il metodo richiama l'omonimo metodo `addStream(MediaStream stream)` della classe `WebRTCAdaptee` che aggiunge lo stream locale alla connessione.
5. Viene successivamente invocato il metodo `createOffer()` della classe `PeerConnection` che richiama a sua volta il metodo `offer()` della classe `WebRTCAdaptee`. L'invocazione del metodo fa si che venga creato un oggetto delle API WebRTC di tipo `RTCSessionDescription` e che questo oggetto venga impostato all'interno dell'oggetto `pc` di tipo `webkitRTCPeerConnection` come descrizione della sessione locale.
6. Viene richiamato il metodo `getOffer()` della classe `CommunicationLogic` dal quale ritorna l'oggetto di tipo `RTCSessionDescription` creato in precedenza codificato in formato JSON.
7. Viene invocato il metodo dell'oggetto `webSocketUser` di tipo `WebSocketUser call(String utente, String offer)` con lo scopo di inviare, tramite il server, l'oggetto `RTCSessionDescription` all'utente che si vuole chiamare.

Postcondizione: l'utente indicato dall'utente autenticato è stato chiamato e gli è stato inviato l'oggetto di tipo `RTCsessionDescription` codificato in formato JSON.

6.2 Ricezione chiamata

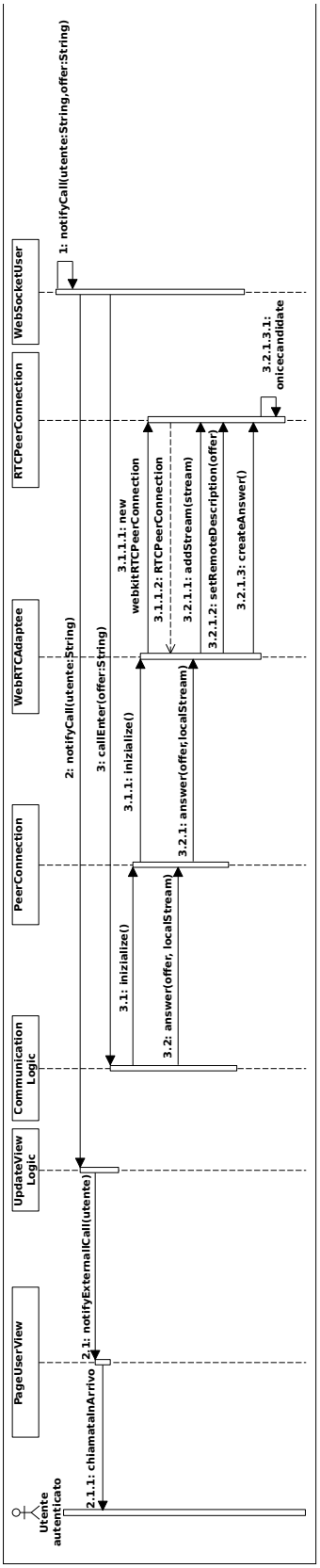

Figura 20: DS 2 - La procedura di notifica di una chiamata in entrata.

Precondizione: l'utente è autenticato, non ci sono comunicazioni in atto.
Descrizione: l'oggetto di tipo `WebSocket` riceve un messaggio in entrata che indica che un altro utente vuole comunicare, vengono quindi eseguiti i seguenti passaggi:

1. Il messaggio viene interpretato dal `WebSocket`, il quale rileva che si tratta di un messaggio che richiede di instaurare una chiamata, estrae quindi dal messaggio il nome dell'utente che ha chiamato e l'oggetto di tipo `RTCSessionDescription` in formato $\text{JSON}_{|g|}$.

2. Vengono invocati i metodi `notifyCall(String utente)` della classe `UpdateViewLogic` passandogli come parametro il nome dell'utente che sta chiamando e `callEnter(String offer)` della classe `CommunicationLogic` passandogli come parametro l'oggetto `RTCSessionDescription` estratto dal messaggio ricevuto.

3. Il metodo `notifyCall(String utente)` precedentemente invocato notifica all'interfaccia grafica la chiamata entrante.

4. Il metodo `callEnter(String offer)` precedentemente invocato richiama il metodo `inizialize()` dell'oggetto di tipo `PeerConnection` che a sua volta richiama l'omonimo metodo della classe `WebRTCAdaptee` che inizializza il campo dati `pc` con un nuovo oggetto di tipo `webkitRTCPeerConnection`.

5. Viene creato un oggetto di tipo `RTCSessionDescription` tramite l'invocazione del metodo `answer(String offer, MediaStream streamLocale)` che richiama, passando tramite opportuni metodi delle classi `PeerConnection` e `WebRTCAdaptee`, i metodi `addStream(MediaStream stream)`, `setRemoteDescription(RTCSessionDescription offer)` e `createAnswer()` delle WebRTC. La chiamata dei metodi delle WebRTC aggiunge la descrizione remota e lo stream locale alla connessione e crea un oggetto da inviare all'utente remoto di tipo `RTCSessionDescription`.

6. Subito dopo la creazione dell'oggetto `RTCSessionDescription` vengono prodotti degli oggetti di tipo `RTCIceCandidate` che vengono salvati all'interno dell'oggetto in formato JSON.

Postcondizione: all'utente autenticato è stato notificata la proposta di un altro utente di comunicare, è stata registrata la sessione remota ed è stata prodotta la sessione locale.

6.3 Accettazione chiamata

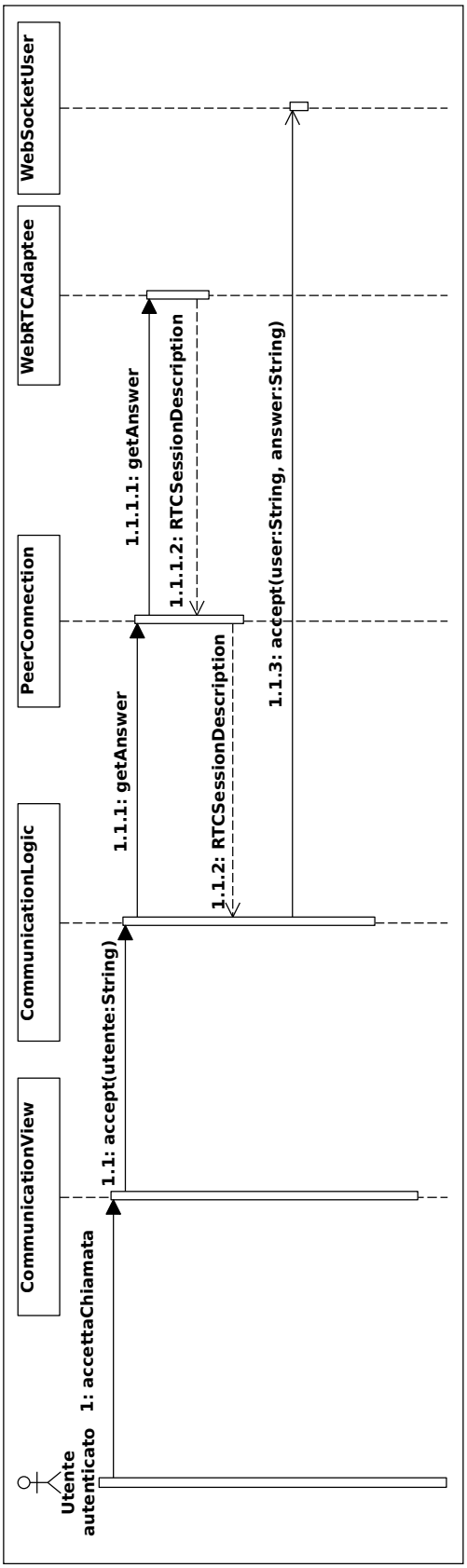

Figura 21: DS 3 - La procedura di accettazione di una chiamata in entrata.

Precondizione: l'utente è autenticato, una chiamata gli è stata appena notificata e viene accettata.
Descrizione: l'utente autenticato accetta la chiamata in arrivo da un utente remoto, vengono eseguiti i seguenti passi:

1. Viene invocato il metodo della classe `CommunicationLogic accept(String utente)` passandogli.
2. Viene recuperata dalla classe `CommunicationLogic` tramite il metodo `getAnswer()` l'oggetto `RTCSessionDescription` che rappresenta poi la descrizione di sessione remota per l'utente remoto.
3. Viene invocato il metodo `accept(String utente, String answer)` della classe `WebSocketUser` al fine di inviare all'utente che ha chiamato la descrizione della sessione.

Postcondizione: l'oggetto `RTCSessionDescription` è stato inviato all'utente che ha chiamato tramite `WebSocket`.

6.4 Chiamata accettata

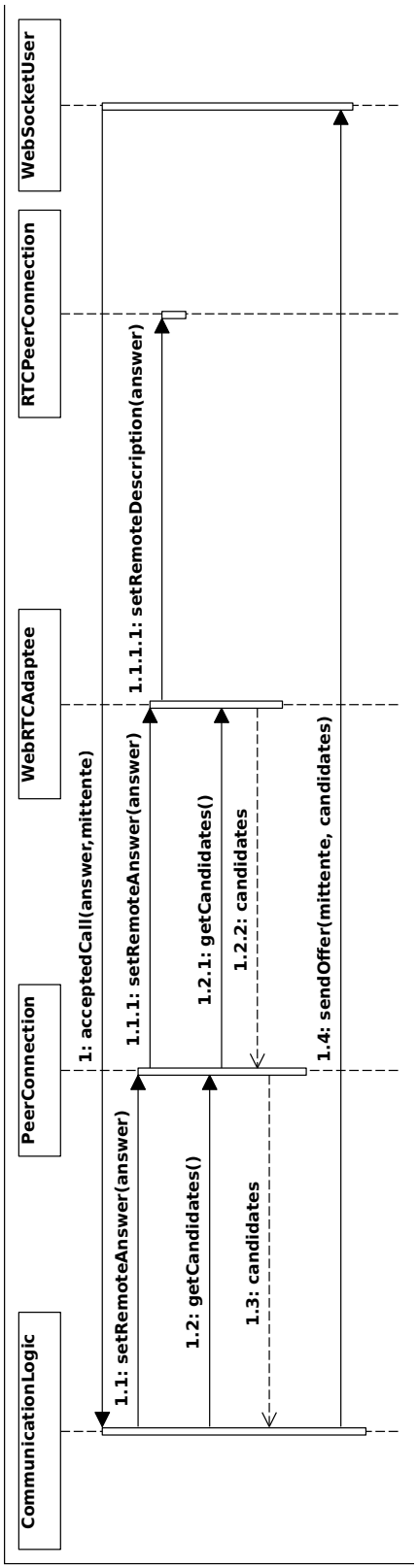

Figura 22: DS 4 - La procedura di scambio degli IceCandidate tra gli utenti durante l'attività di chiamata.

Precondizione: l'utente è autenticato ed una chiamata precedentemente effettuata è stata accettata.
Descrizione: l'oggetto di tipo `WebSocket` riceve un messaggio in entrata che indica che l'utente precedentemente chiamato ha accettato la chiamata, viene estratto dal messaggio ricevuto il nome dell'utente chiamante e l'oggetto di tipo `RTCSessionDescription`, vengono quindi eseguiti i seguenti passaggi;

1. Viene invocato il metodo della classe `CommunicationLogic` `acceptedCall(String utente, String offer)` passandogli come parametri i dati estratti dal messaggio.

2. Viene impostata la descrizione remota rappresentata dall'oggetto `RTCSessionDescription` tramite una chiamata al metodo dell'oggetto `webkitPeerConnection` interno alla classe `WebRTCAdaptee`.

3. Vengono estratti gli oggetti `RTCIceCandidate` in formato JSON creati in precedenza.

4. Vengono inviati gli oggetti `RTCIceCandidate` tramite `WebSocket`,richiamando il metodo `offer(String mittente, String candidate)`.

Postcondizione: gli oggetti `RTCIceCandidate` sono stati inviati all'utente che ha accettato la chiamata.

6.5 Scambio candidati - utente chiamato

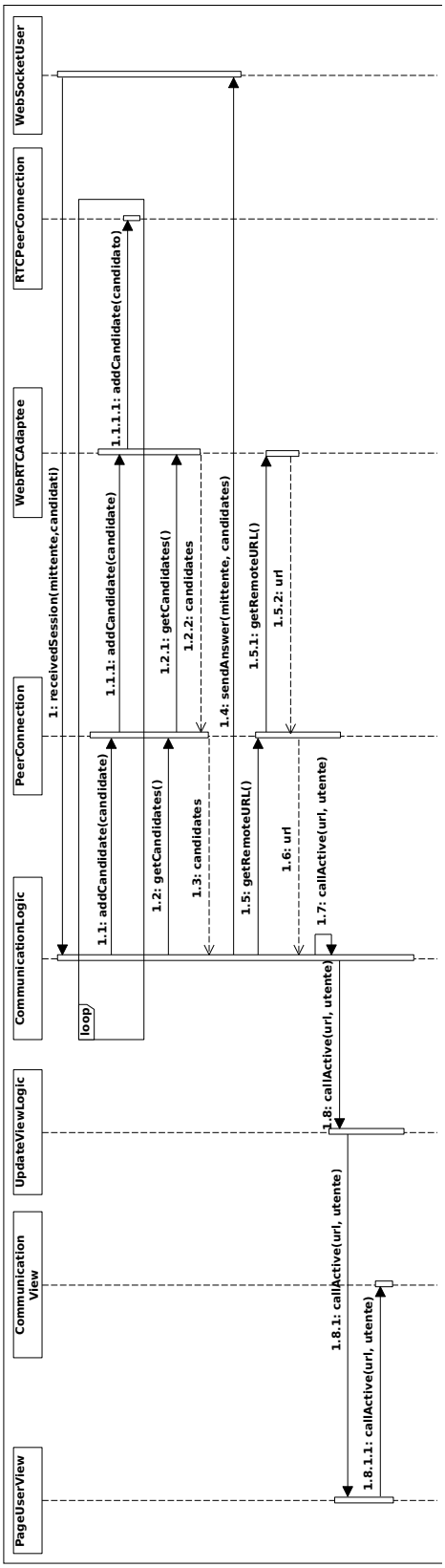

Figura 23: DS 5 - La procedura per l'impostazione e l'invio degli IceCandidate tra gli utenti durante l'attività di chiamata.

Precondizione: l'utente è autenticato ed una chiamata precedentemente ricevuta è stata accettata.
Descrizione: l'oggetto di tipo `WebSocket` riceve un messaggio in entrata. Il messaggio proviene dall'utente che ha precedentemente chiamato e la cui chiamata è già stata accettata, tale messaggio contiene in formato $\text{JSON}_{|g|}$ gli oggetti di tipo `RTCIceCandidate` appartenenti all'utente che ha inviato il messaggio. Vengono quindi eseguiti i seguenti passaggi:

1. Viene invocato il metodo della classe `CommunicationLogic receivedSession(String utente, Vector<String> candidate)` passandogli come parametri i dati estratti dal messaggio.
2. Vengono aggiunti gli oggetti di tipo `RTCIceCandidate` uno alla volta, richiamando ripetutamente il metodo della classe `PeerConnection addCandidate(String candidate)`, che richiama a sua volta il metodo omonimo della classe `WebRTCAdaptee` il quale a sua volta richiama il metodo dell'oggetto `pc` di tipo `webkitRTCPeerConnection addCandidate(RTCIceCandidate)`.
3. Vengono estratti gli oggetti `RTCIceCandidate` in formato JSON creati in precedenza.
4. Vengono inviati gli oggetti `RTCIceCandidate` tramite `WebSocket`, richiamando il metodo `answer(String mittente, String candidate)`.
5. Viene reperito l'url remoto creato all'impostazione della sessione remota.
6. Viene impostato all'interno dell'interfaccia grafica l'url remoto invocando il metodo `callActive(String url)`.

Postcondizione: gli oggetti `RTCIceCandidate` sono stati inviati all'utente che ha accettato la chiamata, l'url remoto è stato impostato.

6.6 Scambio candidati - utente chiamante

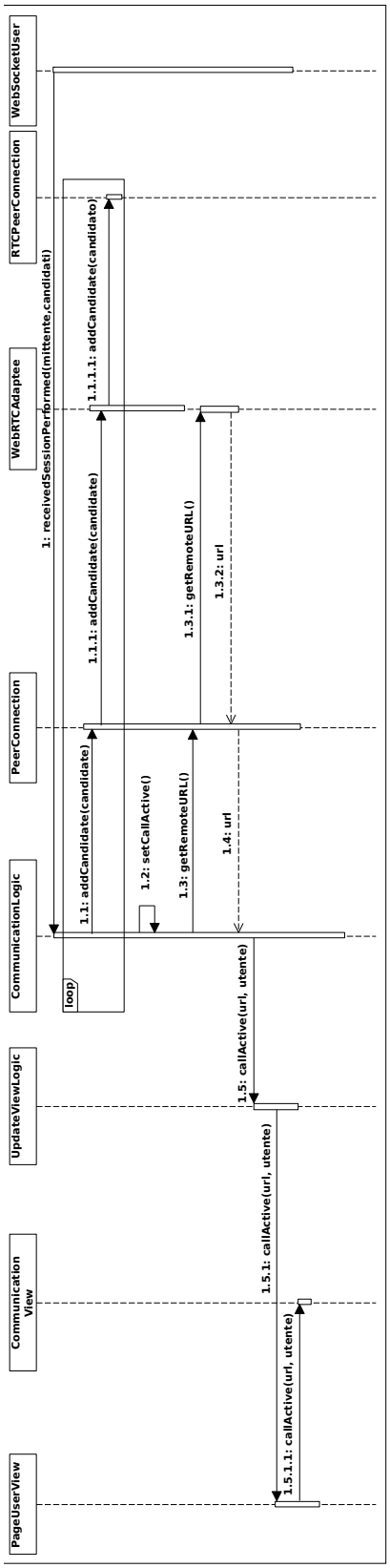

Figura 24: DS 6 - La procedura per l'impostazione degli IceCandidate dell'utente che ha ricevuto la chiamata.

Precondizione: l'utente è autenticato, una chiamata precedentemente effettuata è stata accettata.
Descrizione: l'oggetto di tipo `WebSocket` riceve un messaggio in entrata dall'utente che è stato precedentemente chiamato e che ha accettato la chiamata ed ha inviato in formato $\text{JSON}_{|g|}$ i propri oggetti `RTCIceCandidate`. Vengono quindi eseguiti i seguenti passaggi:

1. Viene invocato il metodo della classe `CommunicationLogic receivedSessionPerformed(String utente, Vector<String> candidate)` passandogli come parametri i dati estratti dal messaggio.
2. Vengono aggiunti gli oggetti di tipo `RTCIceCandidate` uno alla volta, richiamando ripetutamente il metodo della classe `PeerConnection addCandidate(String candidate)`, che richiama a sua volta il metodo omonimo della classe `WebRTCAdaptee` il quale a sua volta richiama il metodo dell'oggetto `pc` di tipo `webkitRTCPeerConnection addCandidate(RTCIceCandidate)`.
3. Viene reperito l'url remoto creato all'impostazione della sessione remota.
4. Viene impostato all'interno dell'interfaccia grafica l'url remoto invocando il metodo `callActive(String url)`.

Postcondizione: l'url remoto è stato impostato, la chiamata ha avuto inizio.

7 Tracciamento

Per il tracciamento relativo a requisiti-componenti si rimanda alla sezione 9 di Specifica Tecnica (*SpecificaTecnica_v3.0.pdf*).

Appendici

A Specifica delle comunicazioni

In questo capitolo vengono specificate le regole adottate nei messaggi di scambio tra gli attori che utilizzano i servizi del programma. Nella comunicazione tra client e server è stato scelto di incapsulare le informazioni da inviare (oggetti $\text{JSON}_{|g|}$ compresi) all'interno di messaggi $\text{XML}_{|g|}$. Gli attori indicati sono:

Client: identifica la parte dell'applicativo utilizzata dall'utente finale. I componenti che lo identificano sono le classi definite nel package `mytalk.client.*`.
Le comunicazioni inviate possono essere riferite ad altri Client, nella fase di instaurare una nuova comunicazione e nella sua gestione, e verso il Server, per ottenere informazioni.

Server: identifica la parte dell'applicativo che gestisce le informazioni e consente la tracciabilità dei client. I componenti che lo identificano sono le classi definite nel package `mytalk.server.*`.
Le comunicazioni inviate possono essere riferite ai Client che si sono registrati nel server.

A.1 Riferimento nomenclatura tag ed attributi

Nei messaggi il nome dei nodi elemento e dei nodi attributo sono stati scelti per consentire una rapida interpretazione e per ridurre la dimensione finale del messaggio. Se ne specifica in seguito il nome e l'ambito di utilizzo.

`c`: Communication.
Dev'essere utilizzato come elemento radice per indicare tutti i messaggi dove si devono gestire informazioni relative alle comunicazioni.
Contiene l'attributo obbligatorio:

`op`: Operation.
Definisce il tipo di operazione da effettuare:

- `am`: Answering Message, aggiungi messaggio di segreteria (vedi A.2.13);
- `amd`: Answering Message Delete, rimuovi messaggi in segreteria (vedi A.2.14);
- `add`: Add, aggiungi statistiche (vedi A.2.10);
- `callNegotation`: Call negotation, informazione per instaurare la comunicazione (vedi A.2.9);
- `callExchange`: Call exchange, informazioni per avviare la comunicazione (vedi A.3.1);
- `find`: Find, effettua una ricerca secondo determinati parametri (vedi A.2.11);
- `stat`: Stat, richiedi informazioni statistiche secondo determinati parametri (vedi A.2.12);
- `userList`: User list, lista utenti registrati (vedi A.2.8).

Può avere gli attributi opzionali:

- `type`: Type, definisce il sottotipo di operazione da effettuare;
- `h`: Hours, definisce il numero di ore.

`d`: Date.
Definisce la data generica nel formato `AAAAMMGGHHMMSS`.

`de`: Date end.
Definisce la data di termine nel formato `AAAAMMGGHHMMSS`.

`ds`: Date start.
Definisce la data di partenza nel formato `AAAAMMGGHHMMSS`.

`dsc`: Description.
Definisce le informazioni aggiuntive.

`er`: Error.
Definisce il commento di errore all'operazione.

`es`: Result.
Definisce il valore risultato dell'operazione.

`g`: Grade.
Definisce il valore di gradimento.

`ip`: IP.
Definisce il valore IP.

`ol`: Online.
Definisce il valore Online.

`m`: E-mail.
Definisce l'indirizzo e-mail.

`msg`: E-mail.
Definisce il pacchetto di informazioni del messaggio di segreteria.

`nm`: Name.
Definisce il nome.

`p`: Password.
Definisce la password.

`pk`: Packages.
Definisce il numero di pacchetti.

`pkL`: Packages lost.
Definisce il numero di pacchetti persi.

`r`: Result.
Elemento che raggruppa le informazioni.

`ref`: Reference.
Definisce il valore di riferimento.

`sc`: Society.
Definisce la società.

`sg`: Sender grade.
Definisce il valore di gradimento del mittente.

`sn`: Surname.
Definisce il cognome.

`ss`: Stream size.
Definisce la dimensione della comunicazione.

`st`: Status.
Definisce lo stato dell'operazione.

`text`: Text.
Definisce il messaggio testuale.

`tl`: Telephone.
Definisce il numero di telefono.

`ud`: UserData.
Dev'essere utilizzato come elemento radice per indicare tutti i messaggi dove si devono gestire informazioni relative all'utente client o amministratore.
Contiene l'attributo obbligatorio:

`op`: Operation.
Specifica l'operazione da effettuare:

- `log`: Login utente (vedi A.2.2);
- `ulg`: Logout utente (vedi A.2.3);
- `add`: Add, aggiungi nuovo utente (vedi A.2.4);
- `udt`: User Data, richiedi dati utente (vedi A.2.5);
- `mod`: Modify, modifica i dati utente (vedi A.2.6);
- `dlt`: Delete, elimina utente (vedi A.2.7).

`ur`: User receiver.
Definisce il riferimento all'utente destinatario.

`us`: User sender.
Definisce il riferimento all'utente mittente.

A.2 Comunicazioni Client-Server

A.2.1 Indirizzo IP

Funzione:
invia all'utente client il suo indirizzo IP.

Messaggio Client -> Server:
Nessuno.

Messaggio Server -> Client:
Messaggio XML

```
<ip>
  {Valore indirizzo IP}
</ip>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.mytalk.com"
 xmlns="http://www.mytalk.com"
 elementFormDefault="qualified">

<xs:element name="ip" type="xs:string" minOccurs="1"
   maxOccurs="1">
</xs:element>

</xs:schema>
```

A.2.2 Login utente

Funzione:
invia al Server le informazioni necessarie per autenticare l'utente.

Messaggio Client -> Server:
Messaggio XML

```
<ud op="log">
  <m>
    {E-mail utente}
  </m>
  <p>
    {Password utente}
  </p>
  <ip>
    {Valore indirizzo IP}
  </ip>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

```

```
<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
      <xs:element name="p" type="xs:string" minOccurs="1
         " maxOccurs="1"/>
      <xs:element name="ip" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<ud op="log">
  <es>
    {Esito operazione}
  </es>
  <m>
    {E-mail utente}
  </m>
  <p>
    {Password utente}
  </p>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="es" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
      <xs:element name="p" type="xs:string" minOccurs="1
         " maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: login effettuato con successo;
- `false`: login non riuscito.

A.2.3 Logout utente

Funzione:
invia al Server le informazioni necessarie per indicare la disconnessione al servizio dell'utente.

Messaggio Client -> Server:
Messaggio XML

```
<ud op="ulg">
  <m>
    {E-mail utente}
  </m>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

```

```
<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Nessuno.

A.2.4 Registrazione nuovo utente

Funzione:
invia al Server le informazioni necessarie registrare un nuovo utente.

Messaggio Client -> Server:
Messaggio XML

```
<ud op="add">
  <m>
    {E-mail utente}
  </m>
  <p>
    {Password utente}
  </p>
  <nm>
    {Nome utente}
  </nm>
  <sn>
    {Cognome utente}
  </sn>
  <sc>
    {Societa' di riferimento per l'utente}
  </sc>
  <tl>
    {Telefono utente}
```

```
  </tl>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
      <xs:element name="p" type="xs:string" minOccurs="1
         " maxOccurs="1"/>
      <xs:element name="nm" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="sn" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="sc" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
      <xs:element name="tl" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<ud op="add">
  <es>
    {Esito operazione}
  </es>
  <er>
```

```
    {Errore nell'operazione}
  </er>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="es" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="er" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: login effettuato con successo;
- `false`: login non riuscito.

Nel caso di esito negativo dell'operazione, l'elemento `er` può contenere un messaggio informativo riguardanti le cause d'insuccesso.

A.2.5 Richiesta dati utente

Funzione:
invia al Server le informazioni per richiedere tutte le informazioni relative ad un specifico utente.

Messaggio Client -> Server:
Messaggio XML

```
<ud op="udt">
  <m>
    {E-mail utente}
  </m>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<ud op="add">
  <m>
    {E-mail utente}
  </m>
  <p>
    {Password utente}
  </p>
  <nm>
    {Nome utente}
  </nm>
  <sn>
    {Cognome utente}
  </sn>
  <sc>
    {Societa' di riferimento per l'utente}
  </sc>
  <tl>
    {Telefono utente}
```

```
  </tl>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
      <xs:element name="p" type="xs:string" minOccurs="1
         " maxOccurs="1"/>
      <xs:element name="nm" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="sn" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="sc" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
      <xs:element name="tl" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

A.2.6 Modifica dati utente

Funzione:
invia al Server le informazioni necessarie per modificare i dati di un utente specifico.

Messaggio Client -> Server:
Messaggio XML

```
<ud op="mod">
  <ref>
    {E-mail utente di riferimento}
  </ref>
  <m>
    {Nuova e-mail utente}
  </m>
  <p>
    {Nuova password utente}
  </p>
  <nm>
    {Nuovo nome utente}
  </nm>
  <sn>
    {Nuovo cognome utente}
  </sn>
  <sc>
    {Nuova societa' di riferimento per l'utente}
  </sc>
  <tl>
    {Nuovo telefono utente}
  </tl>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ref" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="m" type="emailAddress" minOccurs
         ="0" maxOccurs="1"/>
      <xs:element name="p" type="xs:string" minOccurs="0
         " maxOccurs="1"/>
      <xs:element name="nm" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
      <xs:element name="sn" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
      <xs:element name="sc" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
```

```
      <xs:element name="tl" type="xs:string" minOccurs="
         0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<ud op="add">
  <es>
    {Esito operazione}
  </es>
  <er>
    {Errore nell'operazione}
  </er>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="ud">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="es" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="er" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>
```

```
15
16 </xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: modifica effettuata con successo;
- `false`: modifica non riuscita.

Nel caso di esito negativo dell'operazione, l'elemento `er` può contenere un messaggio informativo riguardanti le cause d'insuccesso.

A.2.7 Eliminazione utente

Funzione:
invia al Server le informazioni per l'eliminazione di un specifico utente.

Messaggio Client -> Server:
Messaggio XML

```
1 <ud op="dlt">
2   <m>
3     {E-mail utente}
4   </m>
5 </ud>
```

Validazione XSD

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2 targetNamespace="http://www.mytalk.com"
3 xmlns="http://www.mytalk.com"
4 elementFormDefault="qualified">
5
6 <xs:element name="ud">
7   <xs:complexType>
8     <xs:sequence>
9       <xs:element name="m" type="emailAddress" minOccurs
          ="1" maxOccurs="1"/>
10    </xs:sequence>
11    <xs:attribute name="op" type="xs:string" use="
        required"/>
12  </xs:complexType>
13 </xs:element>
14
15 <xs:simpleType name="emailAddress">
16    <xs:restriction base="xs:string">
17      <xs:pattern value="[^@]+@[^\.]+\..+"/>
18    </xs:restriction>
19 </xs:simpleType>
```

```
20
21 </xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
1 <ud op="add">
2   <es>
3     {Esito operazione}
4   </es>
5   <er>
6     {Errore nell'operazione}
7   </er>
8 </ud>
```

Validazione XSD

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2 targetNamespace="http://www.mytalk.com"
3 xmlns="http://www.mytalk.com"
4 elementFormDefault="qualified">
5
6 <xs:element name="ud">
7   <xs:complexType>
8     <xs:sequence>
9       <xs:element name="es" type="xs:string" minOccurs="
          1" maxOccurs="1"/>
10      <xs:element name="er" type="xs:string" minOccurs="
          1" maxOccurs="1"/>
11    </xs:sequence>
12    <xs:attribute name="op" type="xs:string" use="
        required"/>
13  </xs:complexType>
14 </xs:element>
15
16 </xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: eliminazione effettuata con successo;
- `false`: eliminazione non riuscita.

Nel caso di esito negativo dell'operazione, l'elemento `er` può contenere un messaggio informativo riguardanti le cause d'insuccesso.

A.2.8 Lista utenti

Funzione:
invia al Server la richiesta per ottenere la lista di tutti gli utenti registrati.

Messaggio Client -> Server:
Messaggio XML

```
<c op="userList">
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">


<xs:element name="c">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="op" type="xs:string" use="
           required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>


</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<c op="userList">
  <ud>
    <m>
      {E-mail utente}
    </m>
    <nm>
      {Nome utente}
    </nm>
    <sn>
      {Cognome utente}
    </sn>
    <ol>
      {Utente online}
    </ol>
  </ud>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ud" minOccurs="0" maxOccurs="
           unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="m" type="emailAddress"
                 minOccurs="1" maxOccurs="1"/>
              <xs:element name="nm" type="xs:string"
                 minOccurs="1" maxOccurs="1"/>
              <xs:element name="sn" type="xs:string"
                 minOccurs="1" maxOccurs="1"/>
              <xs:element name="ol" type="xs:string"
                 minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="op" type="xs:string" use="
         required"/>
    </xs:complexType>
  </xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

A.2.9 Negoziazione di chiamata

Funzione:
messaggio inviato al server da parte di un utente mittente con le informazioni necessarie per poter comunicare con un utente destinatario.

Messaggio Client mittente -> Server:
Messaggio XML

```
<c op="callNegotation" type="offer">
  <us>
    {E-mail utente mittente}
  </us>
  <ur>
    {E-mail utente destinatario}
  </ur>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="type" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Client destinatario -> Server:
Messaggio XML

```
<c op="callNegotation" type="offer">
  <us>
    {E-mail utente destinatario}
  </us>
  <ur>
```

```
    {E-mail utente mittente}
  </ur>
  <st>
    {Accettazione della chiamata}
  </st>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="st" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="type" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Il valore contenuto nell'elemento `st` deve essere:

- `accept`: richiesta di comunicazione accettata da parte dell'utente destinatario;
- `refused`: richiesta di comunicazione rifiutata da parte dell'utente destinatario.

Messaggio Server -> Client mittente:
Messaggio XML

```
<c op="callNegotation" type="answer">
  <us>
    {E-mail utente destinatario}
  </us>
  <ur>
    {E-mail utente mittente}
  </ur>
  <st>
    {Accettazione della chiamata}
  </st>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="st" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="type" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Questo messaggio viene inviato dal Server solo nel caso in cui l'utente destinatario non sia attualmente online; in questo caso, il valore dell'elemento `st` deve essere `offline`.

A.2.10 Inserimento nuove statistiche di chiamata

Funzione:
invia al Server le informazioni necessarie per inserire delle nuove statistiche di chiamata.

Messaggio Client -> Server:
Messaggio XML

```
<c op="add">
  <us>
    {E–mail utente mittente}
  </us>
  <ur>
    {E–mail utente destinatario}
  </ur>
  <pk>
    {Numero pacchetti trasmessi}
  </pk>
  <pkL>
    {Numero pacchetti persi}
  </pkL>
  <ss>
    {Numero di byte trasmessi}
  </ss>
  <ds>
    {Data d'inizio}
  </ds>
  <de>
    {Data di fine}
  </de>
  <sg>
    {Indice di gradimento dell'utente mittente}
  </sg>
  <rg>
    {Indice di gradimento dell'utente destinatario}
  </rg>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
```

```
8       <xs:sequence>
9         <xs:element name="us" type="emailAddress"
             minOccurs="1" maxOccurs="1"/>
10        <xs:element name="ur" type="emailAddress"
             minOccurs="1" maxOccurs="1"/>
11        <xs:element name="pk" type="xs:long" minOccurs="1"
              maxOccurs="1"/>
12        <xs:element name="pkL" type="xs:long" minOccurs="1
             " maxOccurs="1"/>
13        <xs:element name="ss" type="xs:long" minOccurs="1"
              maxOccurs="1"/>
14        <xs:element name="ds" type="dateType" minOccurs="1
             " maxOccurs="1"/>
15        <xs:element name="de" type="dateType" minOccurs="1
             " maxOccurs="1"/>
16        <xs:element name="sg" type="grade" minOccurs="1"
             maxOccurs="1"/>
17        <xs:element name="rg" type="grade" minOccurs="1"
             maxOccurs="1"/>
18      </xs:sequence>
19      <xs:attribute name="op" type="xs:string" use="
           required"/>
20      <xs:attribute name="type" type="xs:string" use="
           required"/>
21    </xs:complexType>
22  </xs:element>
23
24  <xs:simpleType name="emailAddress">
25      <xs:restriction base="xs:string">
26        <xs:pattern value="[^@]+@[^\.]+\..+"/>
27      </xs:restriction>
28  </xs:simpleType>
29
30  <xs:simpleType name="dateType">
31     <xs:restriction base="xs:string">
32         <xs:pattern value="\d{14}"/>
33     </xs:restriction>
34  </xs:simpleType>
35
36  <xs:simpleType name="grade">
37     <xs:restriction base="xs:byte">
38         <xs:minInclusive value="0"/>
39         <xs:maxInclusive value="5"/>
40     </xs:restriction>
41  </xs:simpleType>
42
43  </xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<c op="add">
  <es>
    {Esito operazione}
  </es>
  <er>
    {Errore nell'operazione}
  </er>
</ud>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="es" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="er" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: statistiche inserite con successo;
- `false`: inserimento nuove statistiche non riuscito.

Nel caso di esito negativo dell'operazione, l'elemento `er` può contenere un messaggio informativo riguardanti le cause d'insuccesso.

A.2.11 Verifica esistenza utente

Funzione:
invia al Server le informazioni necessarie per verificare se esiste un utente registrato che rispetti i vincoli fissati.

Messaggio Client -> Server:
Messaggio XML

```
<c op="find">
  <ip>
    {Valore indirizzo IP}
  </ip>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ip" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<c op="find">
  <es>
    {Esito operazione}
  </es>
  <m>
    {E-mail utente}
  </m>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
```

```
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="es" type="xs:string" minOccurs="
         1" maxOccurs="1"/>
      <xs:element name="m" type="emailAddress" minOccurs
         ="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Il valore contenuto nell'elemento `es` deve essere:

- `true`: la ricerca ha prodotto qualche risultato;
- `false`: la ricerca non ha prodotto alcun risultato.

A.2.12 Ricerca valori statistici

Funzione:
invia al Server le informazioni necessarie per ottenere delle particolari informazioni relative alle statistiche di comunicazione.

Messaggio Client -> Server:
Messaggio XML

```
<c op="stat" t="{Arco temporale in secondi}">
  <m>
    {E-mail utente}
  </m>
  <g>
    {Indice di gradimento}
  </g>
  <ds>
    {Data d'inizio}
  </ds>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="0" maxOccurs="1"/>
      <xs:element name="g" type="grade" minOccurs="0"
         maxOccurs="1"/>
      <xs:element name="ds" type="dateType" minOccurs="0
         " maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="t" type="xs:nonNegativeInteger"
       use="required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="dateType">
   <xs:restriction base="xs:string">
       <xs:pattern value="\d{14}"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="grade">
   <xs:restriction base="xs:byte">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="5"/>
   </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Almeno uno degli elementi interni deve essere presente.

Messaggio Server -> Client:

Messaggio XML

```
<c op="stat" t="{Arco temporale in secondi}">
  <r>
    <us>
      {E-mail utente mittente}
    </us>
    <ur>
      {E-mail utente destinatario}
    </ur>
    <pk>
      {Numero pacchetti trasmessi}
    </pk>
    <pkL>
      {Numero pacchetti persi}
    </pkL>
    <ss>
      {Numero di byte trasmessi}
    </ss>
    <ds>
      {Data d'inizio}
    </ds>
    <de>
      {Data di fine}
    </de>
    <sg>
      {Indice di gradimento dell'utente mittente}
    </sg>
    <rg>
      {Indice di gradimento dell'utente destinatario}
    </rg>
  </r>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="r" minOccurs="0" maxOccurs="
         unbounded">
        <xs:complexType>
          <xs:sequence>
```

```
12              <xs:element name="us" type="emailAddress"
                   minOccurs="1" maxOccurs="1"/>
13              <xs:element name="ur" type="emailAddress"
                   minOccurs="1" maxOccurs="1"/>
14              <xs:element name="pk" type="xs:long"
                   minOccurs="1" maxOccurs="1"/>
15              <xs:element name="pkL" type="xs:long"
                   minOccurs="1" maxOccurs="1"/>
16              <xs:element name="ss" type="xs:long"
                   minOccurs="1" maxOccurs="1"/>
17              <xs:element name="ds" type="dateType"
                   minOccurs="1" maxOccurs="1"/>
18              <xs:element name="de" type="dateType"
                   minOccurs="1" maxOccurs="1"/>
19              <xs:element name="sg" type="grade" minOccurs
                   ="1" maxOccurs="1"/>
20              <xs:element name="rg" type="grade" minOccurs
                   ="1" maxOccurs="1"/>
21            </xs:sequence>
22          </xs:complexType>
23        </xs:element>
24      </xs:sequence>
25      <xs:attribute name="op" type="xs:string" use="
           required"/>
26      <xs:attribute name="t" type="xs:nonNegativeInteger"
           use="required"/>
27    </xs:complexType>
28  </xs:element>
29
30  <xs:simpleType name="emailAddress">
31    <xs:restriction base="xs:string">
32      <xs:pattern value="[^@]+@[^\.]+\..+"/>
33    </xs:restriction>
34  </xs:simpleType>
35
36  <xs:simpleType name="dateType">
37    <xs:restriction base="xs:string">
38      <xs:pattern value="\d{14}"/>
39    </xs:restriction>
40  </xs:simpleType>
41
42  <xs:simpleType name="grade">
43    <xs:restriction base="xs:byte">
44      <xs:minInclusive value="0"/>
45      <xs:maxInclusive value="5"/>
46    </xs:restriction>
47  </xs:simpleType>
48
```

```
</xs:schema>
```

A.2.13 Aggiunta messaggio di segreteria

Funzione:
invia al Server le informazioni necessarie per lasciare un messaggio testuale in segreteria da far pervenire all'utente destinatario.

Messaggio Client -> Server:
Messaggio XML

```
<c op="am">
  <us>
    {E-mail utente mittente}
  </us>
  <ur>
    {E-mail utente destinatario}
  </ur>
  <ds>
    {Data invio messaggio}
  </ds>
  <msg>
    {Messaggio testuale}
  </msg>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ds" type="dateType" minOccurs="1
         " maxOccurs="1"/>
      <xs:element name="msg" type="xs:string" minOccurs=
         "1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
```

```
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="dateType">
   <xs:restriction base="xs:string">
       <xs:pattern value="\d{14}"/>
   </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Messaggio XML

```
<c op="am">
  <ur>
    {E-mail utente destinatario}
  </ur>
  <msg>
    <us>
      {E-mail utente mittente}
    </us>
    <d>
      {Data d'inizio}
    </d>
    <text>
      {Data d'inizio}
    </text>
  </msg>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
    <xs:complexType>
      <xs:sequence>
```

```
9          <xs:element type="emailAddress" name="ur"/>
10         <xs:element name="msg" minOccurs="1" maxOccurs="
               unbounded">
11           <xs:complexType>
12             <xs:sequence>
13               <xs:element type="emailAddress" name="us"
                     />
14               <xs:element type="dateType" name="d"/>
15               <xs:element type="xs:string" name="text"/>
16             </xs:sequence>
17           </xs:complexType>
18         </xs:element>
19       </xs:sequence>
20       <xs:attribute type="xs:string" name="op"/>
21     </xs:complexType>
22   </xs:element>
23
24 <xs:simpleType name="emailAddress">
25     <xs:restriction base="xs:string">
26       <xs:pattern value="[^@]+@[^\.]+\..+"/>
27     </xs:restriction>
28 </xs:simpleType>
29
30 <xs:simpleType name="dateType">
31    <xs:restriction base="xs:string">
32        <xs:pattern value="\d{14}"/>
33    </xs:restriction>
34 </xs:simpleType>
35
36 </xs:schema>
```

A.2.14 Rimozione messaggi in segreteria

Funzione:
invia al Server le informazioni necessarie eliminare i messaggi presenti in segreteria.

Messaggio Client -> Server:
Messaggio XML

```
1 <c op="amd">
2   <m>
3     {E-mail utente}
4   </m>
5 </c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="m" type="emailAddress" minOccurs
         ="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Server -> Client:
Nessuno.

A.3 Comunicazioni Client-Client

A.3.1 Scambio informazioni di chiamata

Funzione:
messaggio inviato da parte di un utente mittente con le informazioni necessarie per poter scambiare informazioni utili per poter instaurare una comunicazione con un utente destinatario.
Il server ha la funzione di far rimbalzare il messaggio tra i due client.

Messaggio Client Mittente -> Server:
Messaggio XML

```
<c op="callExchange" type="offerDescription">
  <us>
    {E-mail utente mittente}
  </us>
  <ur>
    {E-mail utente destinatario}
  </ur>
  <dsc>
```

```
    {Contenuto informativo}
  </dsc>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="dsc" type="xs:string" minOccurs=
         "1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="type" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Messaggio Client Destinatario -> Server:
Messaggio XML

```
<c op="callExchange" type="offerDescription">
  <us>
    {E-mail utente destinatario}
  </us>
  <ur>
    {E-mail utente mittente}
  </ur>
  <dsc>
```

```
    {Contenuto informativo}
  </dsc>
</c>
```

Validazione XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mytalk.com"
xmlns="http://www.mytalk.com"
elementFormDefault="qualified">

<xs:element name="c">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="us" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="ur" type="emailAddress"
         minOccurs="1" maxOccurs="1"/>
      <xs:element name="dsc" type="xs:string" minOccurs=
         "1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="op" type="xs:string" use="
       required"/>
    <xs:attribute name="type" type="xs:string" use="
       required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="emailAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^@]+@[^\.]+\..+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```