



# Facoltà di Ingegneria - Università di Catania

## Smart Network University Communications

## Software Engineering Projects

Relatore

**Prof. Orazio Tomarchio**

Studenti

**Russo Leandro**  
**Invincibile Daniele**  
**Didomenico Nicola**

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica – DIEEI  
Facoltà di Ingegneria Informatica

06 Marzo 2015



# Indice

## 1. Indice

### 1.1. Prefazione

## 2. Ideazione - Iterazione 1

### 2.1. Iterazione 1: Requisiti

#### 2.2. Iterazione 1: Requisiti - UC1\_RequestConnection

#### 2.3. Iterazione 1: Requisiti - UC2\_AccessRoom

#### 2.4. Iterazione 1: Analisi - UC1\_RequestConnection

#### 2.5. Iterazione 1: Analisi - UC2\_AccessRoom

#### 2.6. Iterazione 1: Progettazione

#### 2.7. Iterazione 1: Progettazione - Class Diagram UC1 e UC2

#### 2.8. Iterazione 1: Progettazione - SSD UC1\_RequestConnection

#### 2.9. Iterazione 1: Progettazione - SSD UC1\_AccessRoom

## 3. Refactoring Iterazione 1



## Indice

- 3.1. Refactoring Iterazione 1: Class Diagram UC1 e UC2
- 3.2. Refactoring Iterazione 1: SSD UC1 e UC2
- 3.3. Refactoring Iterazione 1: Implementazione UC1 e UC2
- 3.4. Refactoring Iterazione 1: Test UC1 e UC2

## 4. Elaborazione - Iterazione 2

- 4.1. Iterazione 2: Requisiti - UC3\_SendPublicMessage
- 4.2. Iterazione 2: Analisi - UC3\_SendPublicMessage
- 4.3. Iterazione 2: Progettazione
- 4.4. Iterazione 2: Progettazione Class Diagram UC3
- 4.5. Iterazione 2: Progettazione - SSD UC3\_SendPublicMessage
- 4.6. Iterazione 2: Implementazione - UC3\_SendPublicMessage

## 5. Elaborazione - Iterazione 3

- 5.1. Iterazione 3: Requisiti - UC4\_SendPrivateMessage



## Indice

- 5.2. Iterazione 3: Analisi - UC4\_SendPrivateMessage
- 5.3. Iterazione 3: Progettazione
- 5.4. Iterazione 3: Progettazione Class Diagram UC4
- 5.5. Iterazione 3: Progettazione - SSD UC4\_SendPrivateMessage
- 5.6. Iterazione 3: Implementazione - UC4\_SendPrivateMessage

## 6. Riferimenti

- 6.1. Bibliografia e Sitografia



## Metodologia Applicata

Per lo sviluppo del processo software descritto a breve, è stata applicata la metodologia dell'Unifiled Process (UP) suggerita durante il corso di studi di Ingegneria del Software.

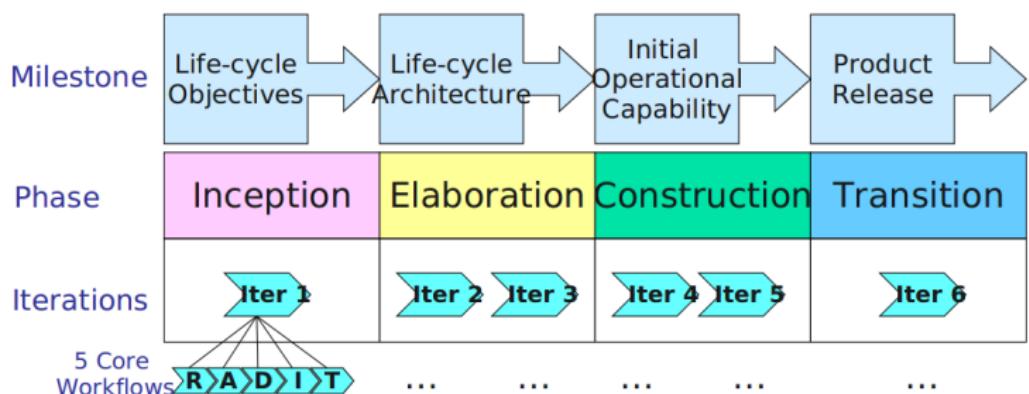


Figura 1 : Software Engineering Process



## Struttura di UP

Il ciclo di vita del progetto è diviso in quattro fasi: Principio (Inception), Elaborazione (Elaboration), Costruzione (Construction) e Transizione (Transition), come si può vedere nella figura 2.



**Figura 2 :** Fasi dell'Unified Process

## Struttura di UP

Ogni fase è formata da una o più iterazioni. Per ogni iterazione si sviluppano in modo iterativo e incrementale 5 attività di lavoro (workflows) RADIT: Requisiti (Requirements), Analisi (Analys), Progettazione (Design), Implementazione (Implementation)e Test, come si può vedere nella figura 3.

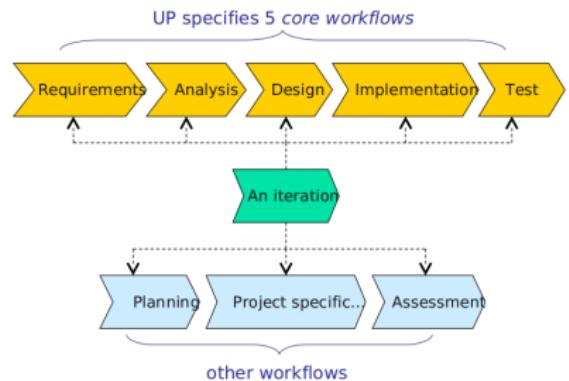


Figura 3 : Iterazioni dell'Unfiled Process



## Documenti del progetto realizzati

Gli elementi che costituiscono il progetto realizzato sono:

- ① Documento di Visione;
- ② Glossario;
- ③ Diagrammi dei casi d'uso;
- ④ Identificazione e descrizione dei casi d'uso in formato breve e dettagliato;
- ⑤ Prototipo UI (User Interface);
- ⑥ Modello, Oggetti e Diagramma di Sequenza di dominio del sistema dei vari casi d'uso in formato dettagliato;
- ⑦ Contratti delle operazioni;
- ⑧ SSD e DCD di progetto del sistema dei vari casi d'uso in formato dettagliato;



## Documenti del progetto realizzati

- ⑨ Codice sorgente dei diagrammi UML realizzati con Astah;
- ⑩ Software del progetto che comprende i casi d'uso in formato dettagliato scritto in linguaggio a oggetti Java, con l'utilizzo di NetBeans IDE 8.0;
- ⑪ Test del software realizzati tramite JUnit 4.0;
- ⑫ Documentazione tramite la Javadoc del codice sorgente;
- ⑬ Guida utente dell'avvio e configurazione del software realizzato;
- ⑭ Documenti di installazione, download dei repository di GitHub;
- ⑮ Documentazione realizzata in L<sup>A</sup>T<sub>E</sub>X.



## Descrizione realtà d'interesse

Nel Piano Nazionale della Ricerca (PNR) messo a punto da HORIZON ITALIA e MIUR viene bandito un progetto chiamato:

**“Smart Network University Communications (SNUC)”** destinato a tutti gli atenei italiani.

Tale progetto è caratterizzato dalle seguenti specifiche descritte nel seguito, che consente ad appassionati, studenti, ricercatori, docenti e imprese di scambiarsi messaggi in tempo reale su condivisioni, integrazioni e competenze di idee per una contaminazione tra ambiti disciplinari diversi, realtà diverse e stimolare nei partecipanti lo sviluppo della cultura dell'intraprendere e dell'innovazione.

In tale sistema si richiedono le seguenti specifiche:



## Descrizione realtà d'interesse

- Esistono diversi canali o stanze virtuali (ciascuna legata a un corso di laurea per ogni facoltà, includendo sia la triennale e la magistrale di quel determinato corso di laurea) nelle quali un utente può entrare per scambiare messaggi con gli altri utenti presenti nella stessa stanza.
- È possibile inviare due tipi di messaggi:
  - ▶ “pubblici”: dei messaggi inviati da un utente e trasmessi a tutti gli altri partecipanti presenti nella stanza;
  - ▶ “privati”: dei messaggi inviati da un utente e trasmessi ad uno specifico partecipante presente nella stessa stanza, in questo caso il destinatario selezionato sarà l’unico a ricevere il messaggio.
- In ogni istante il sistema prevede la presenza di una tipologia di utente particolare, chiamato amministratore. I compiti più importanti di un amministratore sono i seguenti:



## Descrizione realtà d'interesse

- ▶ può creare o eliminare stanze del servizio di messaggistica;
  - ▶ inviare messaggi di avviso ad un utente;
  - ▶ in caso di comportamenti irregolari è possibile espellere un utente dalla stanza e l'utente non può più rientrare fin quando questo non sarà rimosso dalla lista dei partecipanti bannati della stanza presente nel sistema.
- ▶ Il sistema deve mandare dei messaggi di notifica, che permettono di aggiornare l'utente di un cambiamento dello stato del sistema.



## Iterazione 1: Requisiti - Documento di Visione

La caratteristica principale del progetto è quella di consentire ad appassionati, studenti, ricercatori, docenti e imprese di scambiarsi messaggi in tempo reale per ogni ateneo. In particolare in questo progetto sono presenti diverse stanze virtuali che rappresentano le facoltà di un ateneo con due tipologie di utilizzatori di sistema, ovvero l'amministratore e gli utenti del servizio di messaggistica. Con le seguenti caratteristiche:

- l'amministratore gestisce le stanze virtuali e supervisiona gli utenti del servizio;



## Iterazione 1: Requisiti - Documento di Visione

- gli utenti per usufruire di tale servizio inseriscono un nickname e si collegano al server impostando dei parametri di connessione. Ogni utente può accedere a una stanza e inviare messaggi ad ogni utente presente nella stanza in tempo reale, inoltre ognuno può contattare in maniera privata gli altri partecipanti presenti nella stanza.

L'architettura utilizzata per offrire il servizio è di tipo client-server, questo approccio consente di fornire un'interfaccia più flessibile per l'accesso al servizio di messaggistica anche con un semplice browser, senza modificare pesantemente la progettazione rispetto ad un architettura peer-to-peer.

## Iterazione 1: Requisiti - Diagrammi dei casi d'uso

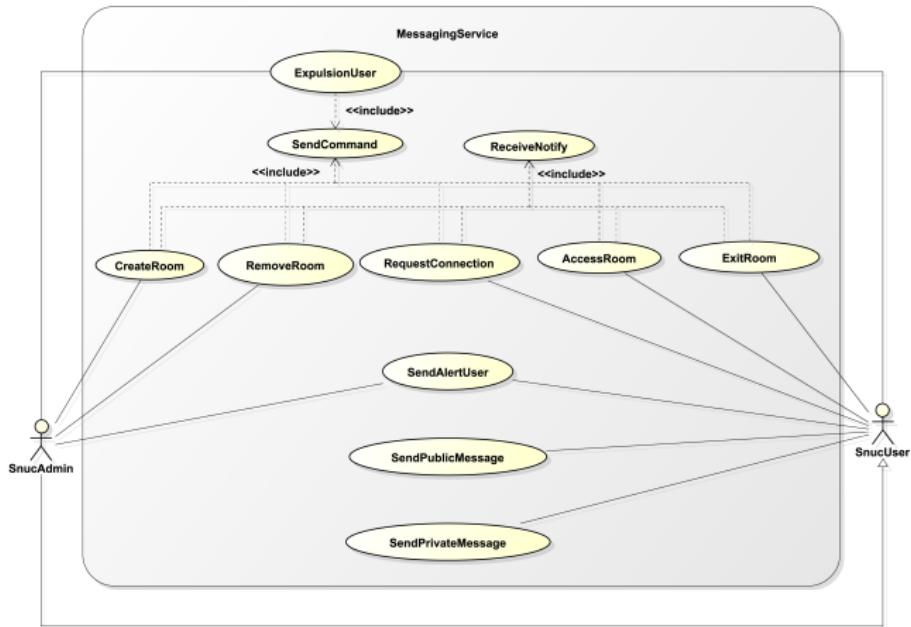


Figura 4 : Diagrammi dei casi uso



## Iterazione 1: Requisiti - Modello dei casi d'uso

Attori Identificati: Amministratore (SnucAdmin) e Utente del Servizio di Messaggistica (SnucUser).

Casi d'uso identificati: RequestConnection, AccessRoom, SendPublicMessage, SendPrivateMessage, ReceiveNotify, SendCommand, ExitRoom, CreateRoom, RemoveRoom, SendAlertUser, ExpulsionUser.

Descrizione breve dei casi d'uso identificati:

- ① **RequestConnection**. L'utente si connette al sistema specificando il nickname, l'indirizzo e la porta del servizio di messaggistica.
- ② **AccessRoom**. L'utente richiede al sistema l'ingresso in una specifica stanza del servizio (che deve essere stata precedentemente creata dall'amministratore del servizio).



## Iterazione 1: Requisiti - Modello dei casi d'uso

- ③ **SendPublicMessage.** Il partecipante al servizio di messaggistica invia un messaggio “pubblico” che viene inviato dal sistema a tutti i partecipanti che si trovano nella stessa stanza di colui che ha inviato il messaggio.
- ④ **SendPrivateMessage.** Il partecipante al servizio di messaggistica invia un messaggio “privato” ad uno specifico partecipante.
- ⑤ **ReceiveNotify.** Il sistema manda dei messaggi di notifica, che permettono di aggiornare l’utente di un cambiamento dello stato del sistema.
- ⑥ **SendCommand.** Il sistema è in grado di ricevere e interpretare dei comandi.



## Iterazione 1: Requisiti - Modello dei casi d'uso

- ⑦ **ExitRoom.** Un partecipante richiede al sistema l'uscita dal servizio. Questo comporta la sua eliminazione dall'insieme dei partecipanti presenti nella stanza del servizio cui era precedentemente associato l'utente.
- ⑧ **CreateRoom.** L'amministratore del servizio di messaggistica è responsabile della creazione (preventiva) delle stanze che potranno successivamente essere visitate dai partecipanti.
- ⑨ **RemoveRoom.** In ogni momento l'amministratore può eliminare una stanza dal servizio (ad esempio perchè non ci sono partecipanti). Ciò comporta l'invio preventivo di un messaggio a tutti i partecipanti eventualmente presenti nella stanza, i quali potranno in seguito richiedere l'ingresso in una nuova stanza del servizio.



## Iterazione 1: Requisiti - Modello dei casi d'uso

- ⑩ **SendAlertUser.** L'amministratore può in ogni momento inviare un messaggio di avviso ad uno specifico partecipante al servizio.
- ⑪ **ExpulsionUser.** L'amministratore può in ogni momento espellere da una stanza uno specifico partecipante.





## Glossario

- ▶ SnucUser, User, Utente: è l'utente del servizio di messaggistica che è in grado di registrarsi a delle stanze, di inviare messaggi pubblici e privati.
- ▶ SnucAdmin, Admin, Amministratore: è il gestore del servizio di messaggistica, che possiede la facoltà di creare e/o eliminare una stanza, di inviare un particolare messaggio di avviso ad un partecipante per un comportamento non corretto ed eventualmente di bannarlo eliminandolo dalla stanza.
- ▶ Messaging Service, Servizio di messaggistica, sistema, chat: sistema che si occupa della gestione.



## Glossario

- ▶ Room, stanza, canale: rappresenta un luogo virtuale, dove gli utenti possono scambiarsi informazioni relative alle tematiche trattate.
- ▶ Messaggio Pubblico: messaggio testuale inviato da un partecipante a tutti gli utenti presenti nella stanza.
- ▶ Messaggio Privato: messaggio testuale inviato da un partecipante ad un particolare partecipante presente nella stanza.
- ▶ Notifica: particolari messaggi di avviso inviati dal server agli utenti che usufruiscono del servizio.



## Iterazione 1: Requisiti - UC1\_RequestConnection

**Tabella 1 :** Descrizione dettagliata: caso d'uso UC1\_RequestConnection

Nome caso d'uso	UC1_RequestConnection
Portata	Applicazione Smart Intelligent University Communications
Livello	Obiettivo Utente
Attore primario	SnucUser
Parti interessate e interessi	SnucUser: vuole collegarsi al servizio di messaggistica
Pre-condizioni	L'utente ha bisogno di una connessione di rete
Post-condizioni (garanzia di successo)	L'utente è inserito tra gli utenti online con il nickname confermato dal servizio di messaggistica ottenendo un messaggio di benvenuto
Scenario principale di successo	<ul style="list-style-type: none"> <li>① L'utente inserisci un nickname, l'address e la porta del server.</li> <li>② Il sistema esamina il nickname inviato dall'utente e verifica se è presente una omonimia.</li> <li>③ Il sistema conferma l'inserimento tra gli utenti online inviando una notifica di benvenuto.</li> </ul>



## Iterazione 1: Requisiti - UC1\_RequestConnection

Estensioni (o flussi alternativi)	<p>1A - SnucUser inserisce parametri errati:</p> <ul style="list-style-type: none"> <li>▶ Viene visualizzato un messaggio di errore e viene richiesto nuovamente l'inserimento di tali parametri.</li> </ul> <p>2A - Omonimia del nickname:</p> <ul style="list-style-type: none"> <li>▶ Il sistema cambia il nickname aggiungendo _ al nickname (es. _nickname).</li> <li>▶ Il sistema conferma l'inserimento tra gli utenti online inviando una notifica di benvenuto.</li> </ul>
Requisiti speciali (Requisiti Non Funzionali)	Comunicazione asincrona in cui lo scambio di informazioni avviene in tempo reale, senza sensibili pause tra invio e ricezione del messaggio.
Elenco delle varianti tecnologiche	L'applicazione dovrebbe essere flessibile al funzionamento di diversi protocolli di comunicazione (es. TCP, UDP) e con diversi strati middleware (es. Socket, RMI)
Frequenza di ripetizione	Potrebbe essere quasi ininterrotta
Varie e/o Problemi Aperti	//



## Iterazione 1: Requisiti - UC2\_AccessRoom

**Tabella 2 :** Caso d'uso UC2\_AccessRoom

Nome caso d'uso	UC2_AccessRoom
Portata	Applicazione Smart Intelligent University Communications
Livello	Obiettivo Utente
Attore primario	SnucUser
Parti interessate e interessi	SnucUser: vuole registrarsi e effettuare l'ingresso in una stanza presente nel servizio di messaggistica
Pre-condizioni	Nel sistema è presente almeno una stanza creata da un amministratore.
Post-condizioni (garanzia di successo)	Nel caso di svolgimento normale l'utente è registrato ed è presente nell'insieme degli utenti della stanza specificata.



## Iterazione 1: Requisiti - UC2\_AccessRoom

Scenario principale di successo	<ul style="list-style-type: none"> <li>① L'utente richiede una lista di stanze presenti nel sistema di messaggistica.</li> <li>② Il sistema invia la lista delle stanze.</li> <li>③ L'utente seleziona la stanza tra quelle presenti in lista.</li> <li>④ Il sistema registra l'utente alla stanza.</li> <li>⑤ Il sistema visualizza a video gli utenti presenti nella stanza.</li> <li>⑥ Il sistema visualizza un'area pubblica dove vengono mostrati tutte le conversazioni in corso dal quel momento in poi.</li> </ul>
Estensioni (o flussi alternativi)	3A - Il SnucUser inserisce una stanza non in elenco: <ul style="list-style-type: none"> <li>① Il sistema invia un messaggio di errore.</li> </ul>
Requisiti speciali (Requisiti Non Funzionali)	Comunicazione asincrona in cui lo scambio di informazioni avviene in tempo reale, senza sensibili pause tra invio e ricezione del messaggio.
Elenco delle varianti tecnologiche	L'applicazione dovrebbe essere flessibile al funzionamento di diversi protocolli di comunicazione (es. TCP, UDP) e con diversi strati middleware (es. Socket, RMI)
Frequenza di ripetizione	Potrebbe essere quasi ininterrotta
Varie e/o Problemi Aperti	//



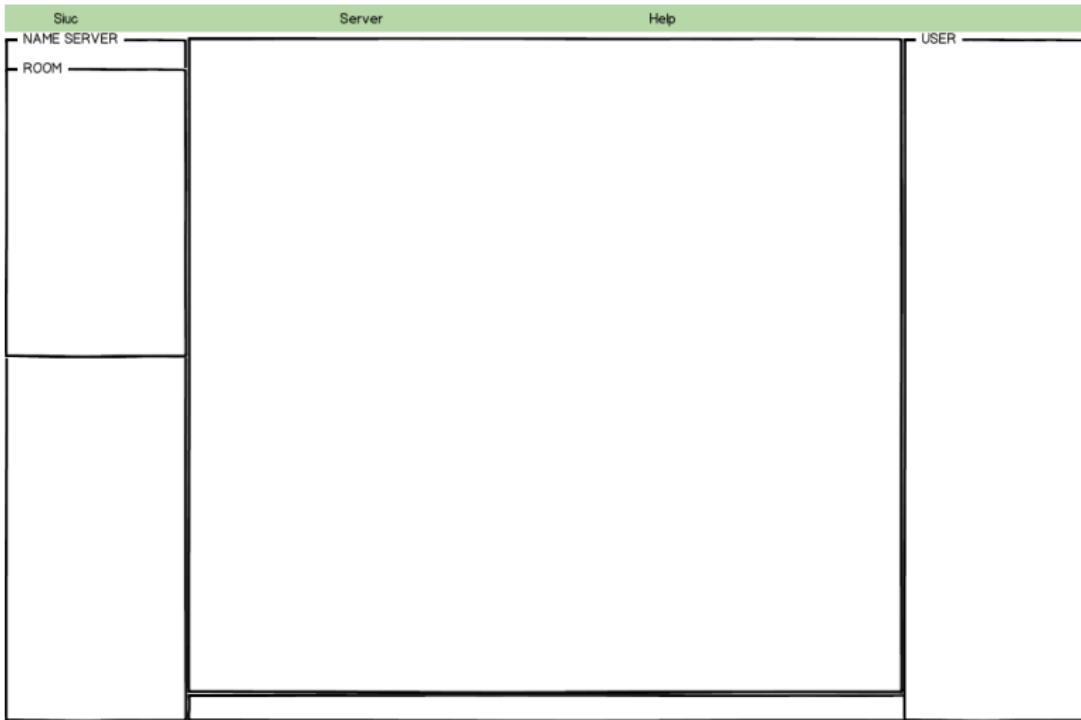
## Descrizione Prototipo User Interface

Viene presentata un prototipo dell'interfaccia grafica (GUI) al cliente realizzata tramite Balsamiq Mockups per rendere più visibile il confine del sistema di iterazione da parte delle figure coinvolte con l'ipotetico sistema da realizzare, in modo da sollevare ipotetiche situazioni problematiche relative.



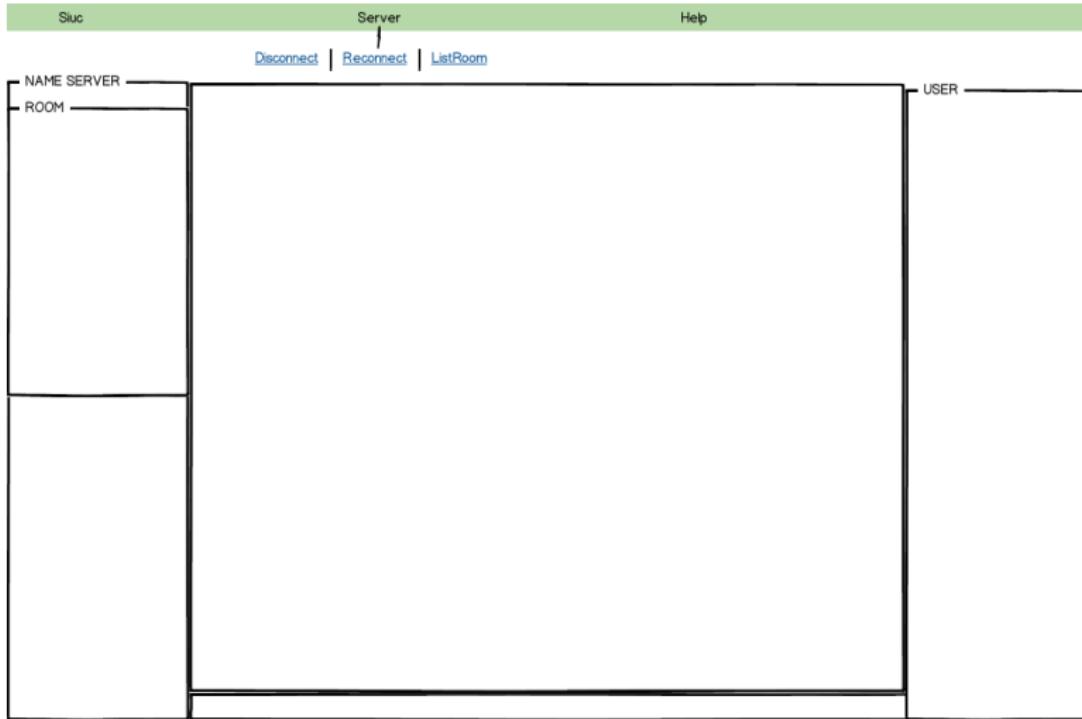


# Iterazione 1: Requisiti - Prototipo User Interface



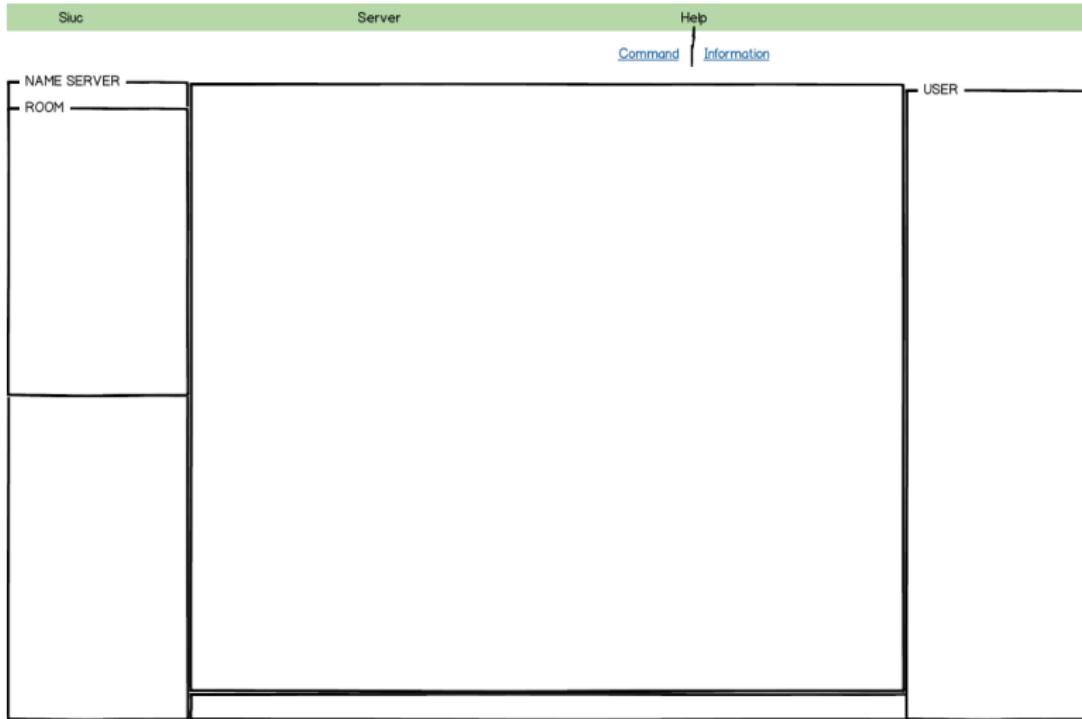


# Iterazione 1: Requisiti - Prototipo User Interface



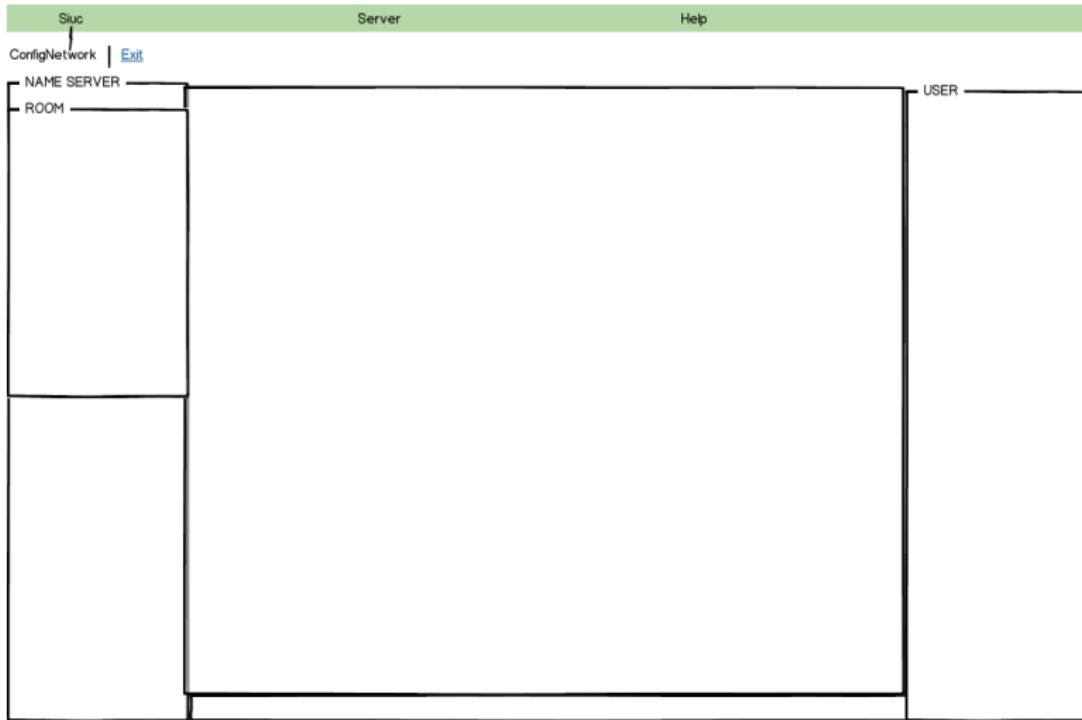


# Iterazione 1: Requisiti - Prototipo User Interface





# Iterazione 1: Requisiti - Prototipo User Interface



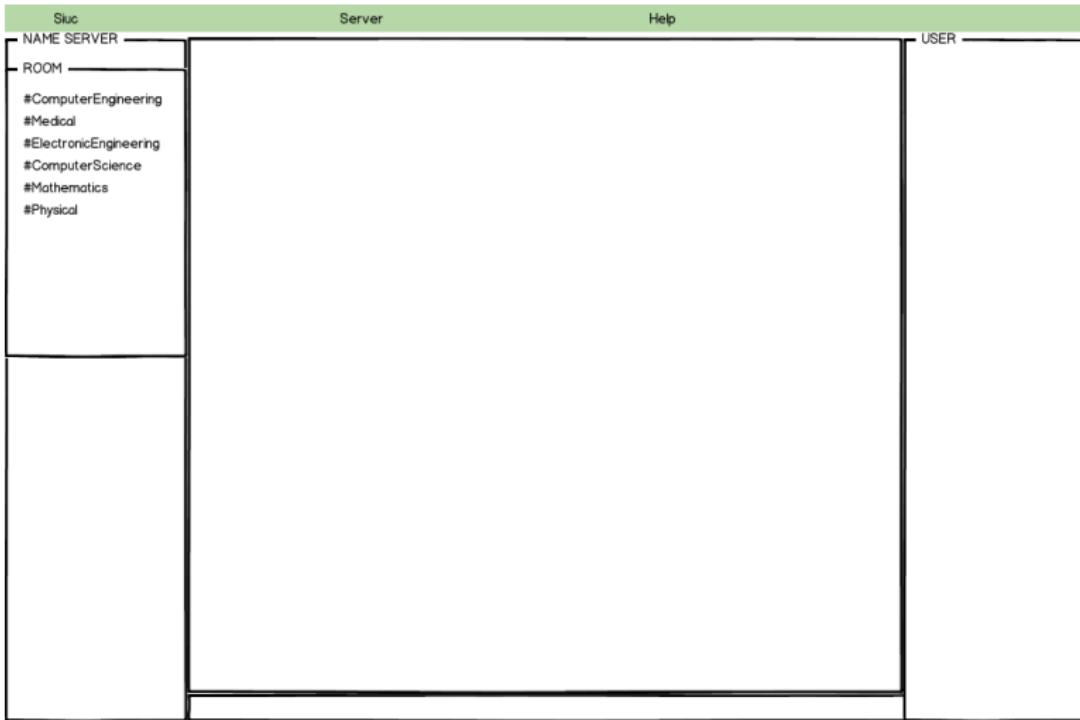


# Iterazione 1: Requisiti - Prototipo User Interface

The diagram illustrates a user interface mockup for Iteration 1. At the top, a green header bar contains the text "Sluc", "ConfigNetwork", and "Exit". To the right of the header are tabs for "Server" and "Help". Below the header, there are three main sections: "NAME SERVER" on the left, "Network" in the center, and "USER" on the right. The "NAME SERVER" section has two dropdown menus: "NAME SERVER" and "ROOM". The "Network" section is a dialog box titled "Network" with fields for "User" (Nick name, Real name) and "Server" (Name, Address, Port). It includes "Close" and "Connect" buttons. The "USER" section is a large empty rectangular area.



# Iterazione 1: Requisiti - Prototipo User Interface





# Iterazione 1: Requisiti - Prototipo User Interface

Sluc	Server	Help
NAME SERVER - ROOM #ComputerEngineering #Medical #ElectronicEngineering #ComputerScience #Mathematics #Physical	<b>19 Gennaio 2015</b> <u>[ora:minuti:secondi]</u> nicola : ciao <u>[ora:minuti:secondi]</u> leandro : funziona ? <u>[ora:minuti:secondi]</u> daniele : bellissima ..... <u>[ora:minuti:secondi]</u> orazio : è un ottimo modello UP...	USER o nicola - leandro - daniele - orazio



# Iterazione 1: Requisiti - Prototipo User Interface

Siuc      Server      Help

NAME SERVER

- ROOM
- #ComputerEngineering**
- #Medical
- #ElectronicEngineering
- #ComputerScience
- #Mathematics
- #Physical

Private Chat

04 Febbraio 2015

[ora:minuti:secondi] nicola : ciao  
 [ora:minuti:secondi] leandro : ciao  
 [ora:minuti:secondi] nicola : riusciremo nell'impresa ?  
 [ora:minuti:secondi] leandro : cettu, cettu, w sant'agata !!!  
 [ora:minuti:secondi] nicola : no w san michele !!!

USER

- o nicola**
- leandro
- daniele
- orazio



# Iterazione 1: Requisiti - Prototipo User Interface

Sluc	Server	Help
NAME SERVER - ROOM #ComputerEngineering <b>#Medical</b> #ElectronicEngineering #ComputerScience #Mathematics #Physical	<p align="center"><b>19 Gennaio 2015</b></p> <p>[ora:minuti:secondi] user_medical 1 : ragazzi questi informatici ne sanno una più del dia'            [ora:minuti:secondi] user_medical 2 : si si...            [ora:minuti:secondi] user_medical 3 : assurdo ....            [ora:minuti:secondi] user_medical 4 : oggi c'è la lezione di anatomia ??</p>	USER <ul style="list-style-type: none"> <li>o user_medical 1</li> <li>- user_medical 2</li> <li>- user_medical 3</li> <li>- user_medical 4</li> </ul>





## Descrizione Analisi - UC1\_RequestConnection

In questa iterazione, del caso d'uso UC1 è di interesse lo scenario principale di successo. Da esso è possibile identificare le seguenti classi concettuali:

- ▶ **User**: rappresenta il generico utente connesso al servizio di messaggistica. È caratterizzato da un “nickname” e può ricevere notifiche dal sistema centrale.
- ▶ **MessagingService**: rappresenta ed incapsula il servizio di messaggistica nel suo complesso. Mantiene una lista di utenti connessi a tale sistema.
- ▶ **Message**: individua un generico messaggio scambiato tra utenti della chat o tra servizio di messaggistica e utente. È costituito da un “content” (contenuto del messaggio), da una “date” (rappresenta la data) e dal “sender” (mittente).



## Descrizione Analisi - UC1\_RequestConnection

- ▶ **Notify:** è una specializzazione del tipo Message ed è caratterizzata da un “typeNotify” che serve a distinguere il tipo di notifica (ad es. CONNECTION\_ACCEPT nel caso in cui la connessione è stata stabilita correttamente).



## Iterazione 1: Analisi - UC1\_RequestConnection

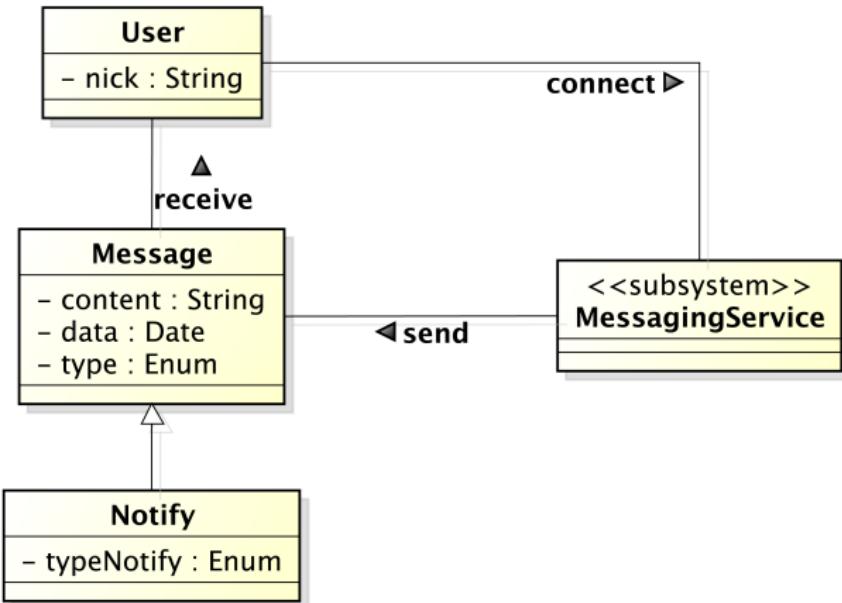


Figura 5 : UC1 - Modello di dominio

## Iterazione 1: Analisi - UC1\_RequestConnection

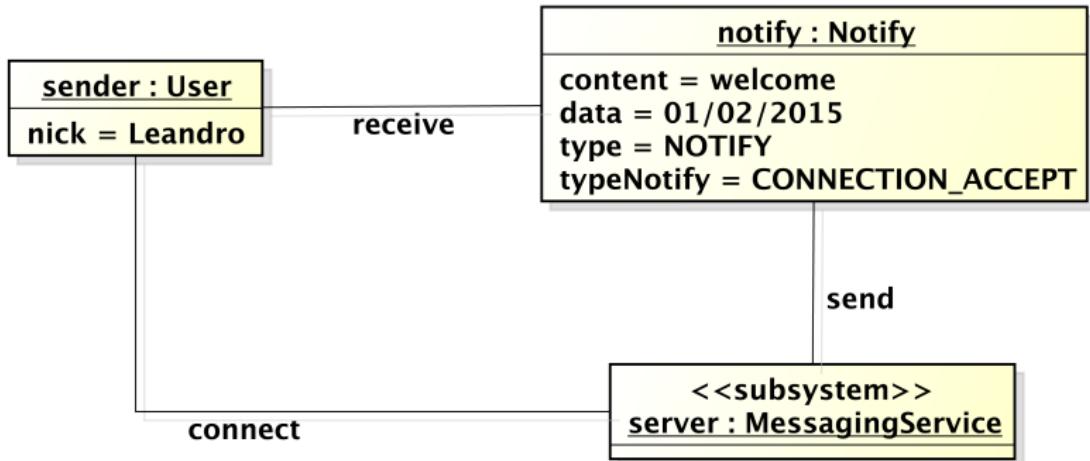


Figura 6 : UC1 - Oggetti di dominio

## Iterazione 1: Analisi - UC1\_RequestConnection

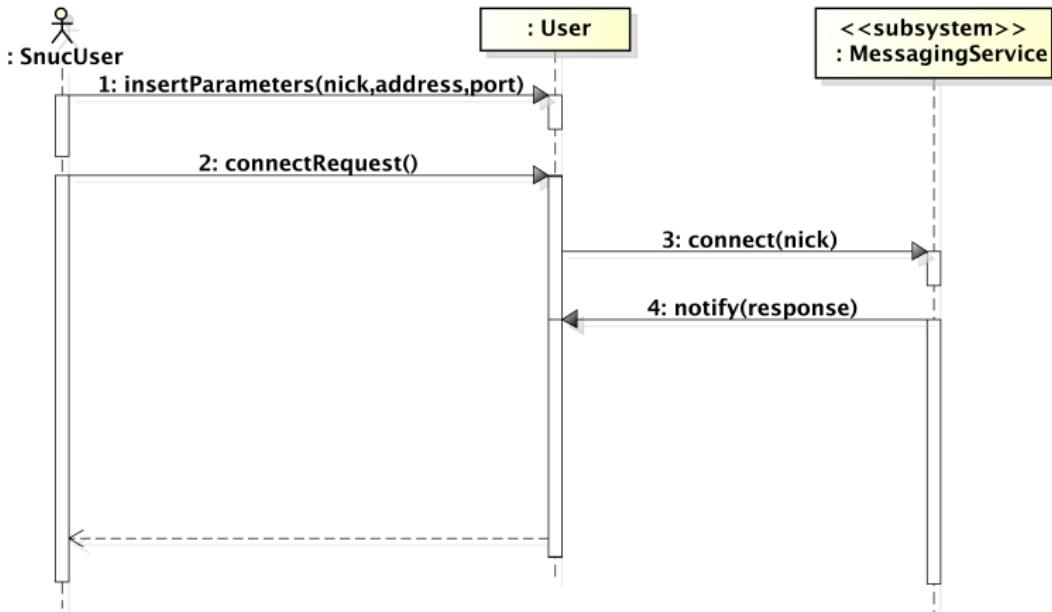


Figura 7 : UC1 - Diagramma di sequenza di sistema



## Iterazione 1: Analisi - UC1 contratti CO1/CO2

**Tabella 3 :** UC1 Contratto CO1 - connect

Operazione	<i>connect(nick: String)</i>
Riferimenti	Caso d'uso: UC1_RequestConnection
Pre-condizione	L'utente ha inserito correttamente i parametri di connessione ("address" e "port")
Post-condizione	L'utente è connesso al servizio di messaggistica e viene aggiunto nella lista degli utenti online

**Tabella 4 :** UC1 Contratto CO2 - notify

Operazione	<i>notify(n: Notify)</i>
Riferimenti	Caso d'uso: UC1_RequestConnection
Pre-condizione	L'utente è connesso al servizio di messaggistica
Post-condizione	L'utente riceve la notifica



## Descrizione Analisi - UC2\_AccessRoom

In questa iterazione, del caso d'uso UC2 è di interesse lo scenario principale di successo. Da esso è possibile identificare le seguenti classi concettuali:

- ▶ **User:** rappresenta il generico utente, caratterizzato da un “nickname”, connesso al servizio di messaggistica. *Può richiedere la lista delle stanze, ricevere notifiche dal sistema centrale. Interagisce con il MessagingService richiedendo la registrazione e l'ingresso in una specifica stanza.*
- ▶ **MessagingService:** rappresenta ed incapsula il servizio di messaggistica nel suo complesso. Mantiene una lista di utenti connessi a tale sistema. *Mantiene una lista di stanze e riceve tramite comandi richieste di ingresso da parte degli utenti.*



## Descrizione Analisi - UC2\_AccessRoom

- ▶ **Message:** individua un generico messaggio scambiato tra utenti della chat o tra servizio di messaggistica e utente. È costituito da un “content” (contenuto del messaggio), da una “date” (rappresenta la data) e dal “sender” (mittente).
- ▶ **Notify:** è una specializzazione del tipo Message ed è caratterizzata da un typeNotify che serve a distinguere il tipo di notifica (ad es. CONNECTIO\_ACCEPT nel caso in cui la connessione è stata stabilita correttamente, *BAD\_COMMAND nel caso in cui il comando inviato dall'User non sia riconosciuto dal Server*).



## Descrizione Analisi - UC2\_AccessRoom

- ▶ **PublicNotify:** è una specializzazione di *Notify* e questo tipo di notifica viene ricevuta da tutti gli utenti registrati alla relativa stanza. Un esempio di *PublicNotify* è la notifica caratterizzata dal seguente *typeNotify*: *UPDATE\_LIST\_USERS*, grazie alla quale viene aggiornata la lista degli utenti registrati nella relativa stanza.
- ▶ **Command:** è una specializzazione di *Message* e rappresenta il comando che viene inviato dall'*User* e ricevuto ed interpretato dal *MessagingService* (es. */join '#Medical'* richiesta da parte dell'utente a registrarsi alla stanza *Medical*).
- ▶ **Room:** è caratterizzata da un nome. Ciascuna istanza individua una specifica stanza nella chat.



## Descrizione Analisi - UC2\_AccessRoom

- ▶ **Register:** mantiene un riferimento all'insieme di partecipanti che in un certo istante sono presenti nella stanza.



## Iterazione 1: Analisi - UC2\_AccessRoom

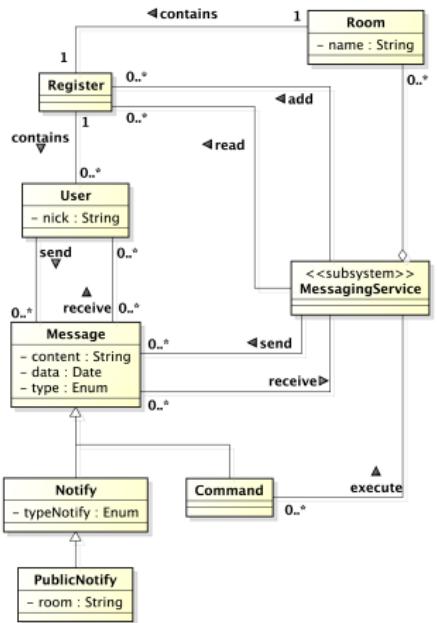
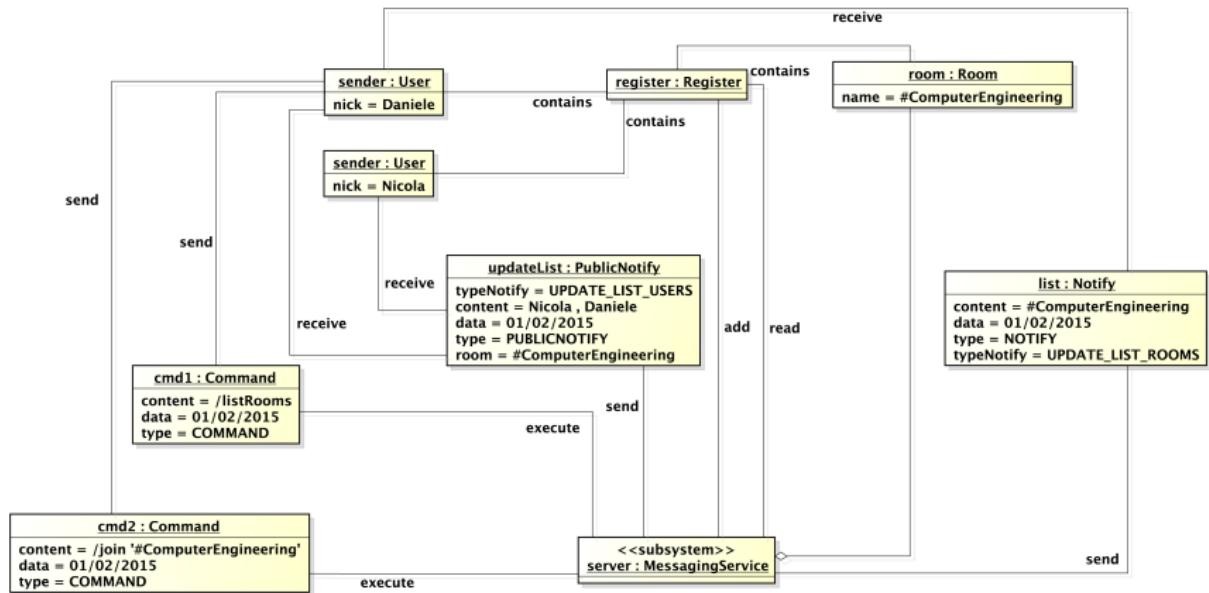


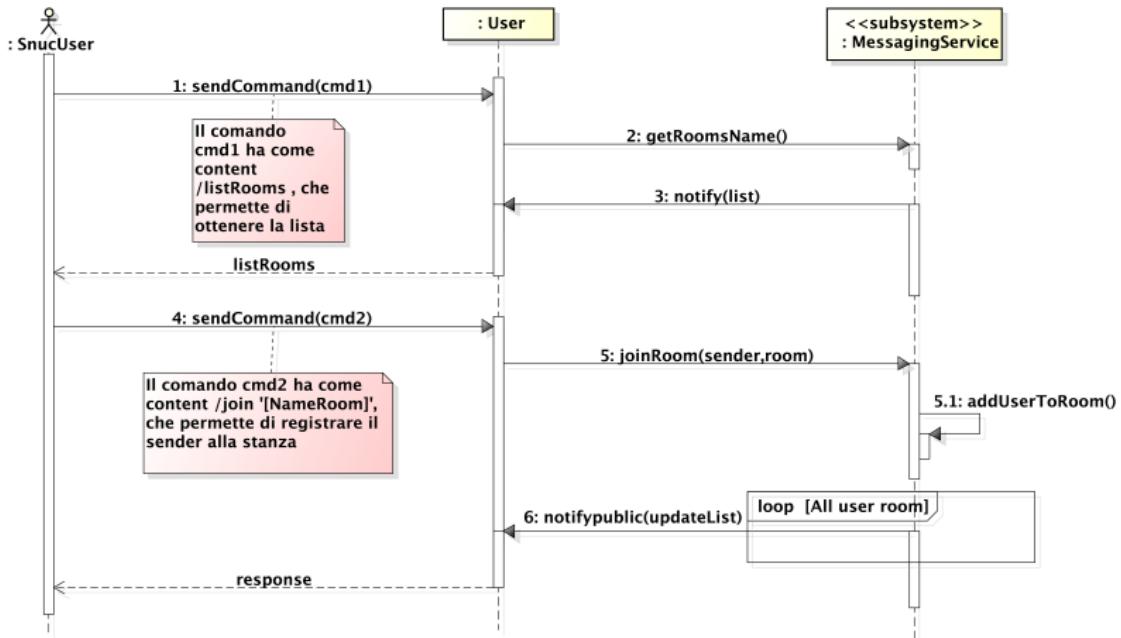
Figura 8 : UC2 - Modello di dominio

## Iterazione 1: Analisi - UC2\_AccessRoom



**Figura 9 :** UC2 - Oggetti di dominio

## Iterazione 1: Analisi - UC2\_AccessRoom



**Figura 10 :** UC2 - Diagramma di sequenza di sistema



## Iterazione 1: Analisi - UC2 contratti CO3/CO4

**Tabella 5 :** UC2 Contratto CO3 - joinRoom

Operazione	<i>joinRoom(sender: String, room: String)</i>
Riferimenti	Caso d'uso: UC2_AccessRoom
Pre-condizione	<ul style="list-style-type: none"> <li>▶ L'utente è connesso al servizio di messaggistica</li> <li>▶ L'utente ha inserito il comando relativo alla registrazione nella stanza</li> </ul>
Post-condizione	Il server interpreta il comando ed inserirà l'utente nella lista degli utenti registrati nella stanza



## Iterazione 1: Analisi - UC2 contratti CO3/CO4

**Tabella 6 :** UC2 Contratto CO4 - notifypublic

Operazione	<i>notifypublic(n: PublicNotify)</i>
Riferimenti	Caso d'uso: UC1_RequestConnection
Pre-condizione	Registrazione utente tra gli utenti della stanza
Post-condizione	Tutti gli utenti registrati nella stanza ricevono la notifica pubblica relativa all'aggiornamento della lista degli utenti



## Descrizione Iterazione 1: Progettazione

### PACKAGE COMMON

- ▶ **Message:** è una classe astratta che definisce tutti gli attributi di un generico messaggio che saranno necessari per la corretta gestione.
- ▶ **Command:** è una sottoclassificazione di Message e permette di identificare che nel contenuto del messaggio avremo un comando.
- ▶ **Notify:** è una sottoclassificazione di Message che permette di identificare le notifiche inviate dal MessagingServer agli User per evidenziare che vi sono stati dei cambiamenti di stato del sistema. Aggiunge un ulteriore attributo per permettere di distinguere il tipo di notifica.



## Descrizione Iterazione 1: Progettazione

- ▶ **PublicNotify:** è una sottoclasse di Notify. Questo tipo di notifica viene inviata dal MessagingService a tutti gli utenti che sono registrati in una determinata stanza. È stato dunque necessario aggiungere un ulteriore attributo in cui viene memorizzata la stanza a cui è riferita. Inoltre in quanto l'utente può essere connesso a più stanze contemporaneamente, l'aggiunta di tale attributo ha permesso all'Utente che riceve una Public-Notify di comprendere a quale stanza è riferita.

### PACKAGE SNUC

- ▶ **UserView:** rappresenta l'interfaccia utente. Implementa l'interfaccia UserInteraction.



## Descrizione Iterazione 1: Progettazione

- ▶ **UserController:** è il primo oggetto oltre allo strato UI a ricevere e coordinare una operazione di sistema. Fa in modo che l'interfaccia utente non contenga logica applicativa.
- ▶ **User:** è l'oggetto dove sono memorizzati tutti i dati relativi all'utente.
- ▶ **UserConnection:** questa classe opera lato Client e si occupa dell' instaurazione della connessione con il servizio di messaggistica, dell'invio e della ricezione di oggetti di tipo Message. In base al tipo di Message ricevuto si occupa di andare a richiamare l'opportuno metodo dell'UserController per la gestione di tale messaggio. Poiché dovrà effettuare la continua ricezione di messaggi è stato necessario lanciare un nuovo Thread che



## Descrizione Iterazione 1: Progettazione

resterà in ascolto per ricevere eventuali messaggi da parte del server.

- ▶ **SnucMain:** tale classe si occupa della creazione e dell'avvio di tutti gli oggetti necessari al funzionamento dell'applicazione lato Client.

### PACKAGE SNUCSERVER

- ▶ **MessagingService:** tale classe che rappresenta il servizio di messaggistica ed in cui sono implementate tutte le funzioni necessarie al funzionamento di tale servizio tra cui per esempio la gestione dell'invio delle notifiche agli User, dell'interpretazione dei comandi e del controllo della disponibilità del nickname. Questa classe lancia inoltre un Thread che si occupa di gestire le connessioni da parte degli User.



## Descrizione Iterazione 1: Progettazione

- ▶ **UserConnectionHandler:** una volta che il MessagingService instaura delle connessioni crea degli oggetti di questo tipo e da quel momento in poi la comunicazione di rete con il corrispettivo Client sarà gestita da tale oggetto. Si occupa in particolare della ricezione di oggetti di tipo Message e di richiamare l'opportuna procedura del server per la loro gestione. Il MessagingService instanzia un UserConnectionHandler per ogni User che si connette. Questo oggetto, per rimanere in continuo ascolto di richieste da parte del client, lancia un Thread che effettua tale operazione.



## Descrizione Iterazione 1: Progettazione

- ▶ **Room:** rappresenta la stanza virtuale dove gli user possono registrarsi. Ha quindi un riferimento alla lista degli utenti che sono registrati e implementa tutti i metodi necessari per la sua gestione.
- ▶ **CommandParser:** si occupa di analizzare i comandi inviati dagli User. Effettua una controllo sulla sintassi e fornisce dei metodi utili per prelevare il comando e i suoi eventuali parametri. Non si occupa però dell'interpretazione del comando in quanto tale funzione sarà gestita dal MessagingService.

# Iterazione 1: Progettazione - Class Diagram Common UC1 e UC2

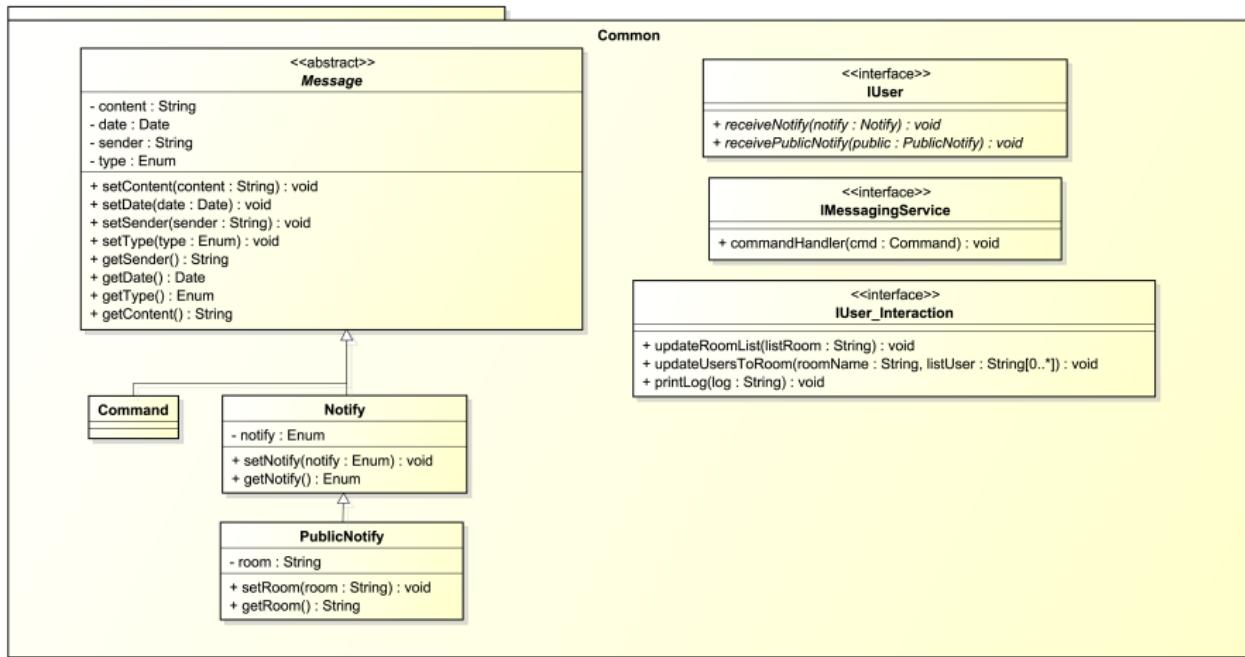


Figura 11 : DCD - Diagramma delle Classi: Package Common

# Iterazione 1: Progettazione - Class Diagram Snuc Server UC1 e UC2

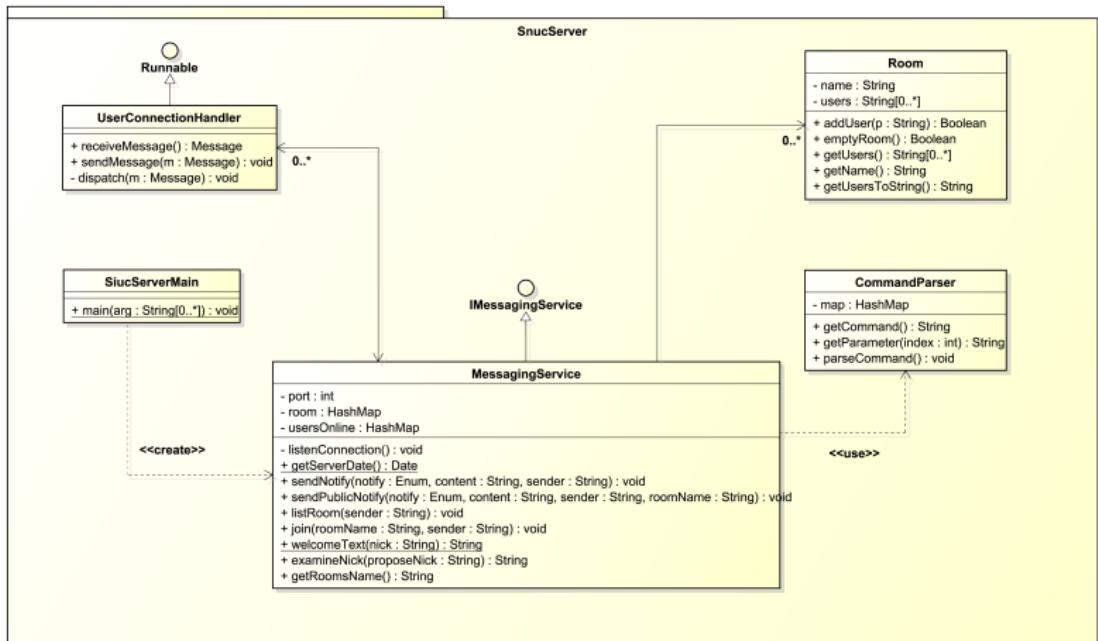


Figura 12 : DCD - Diagramma delle Classi: Package Snuc Server

## Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

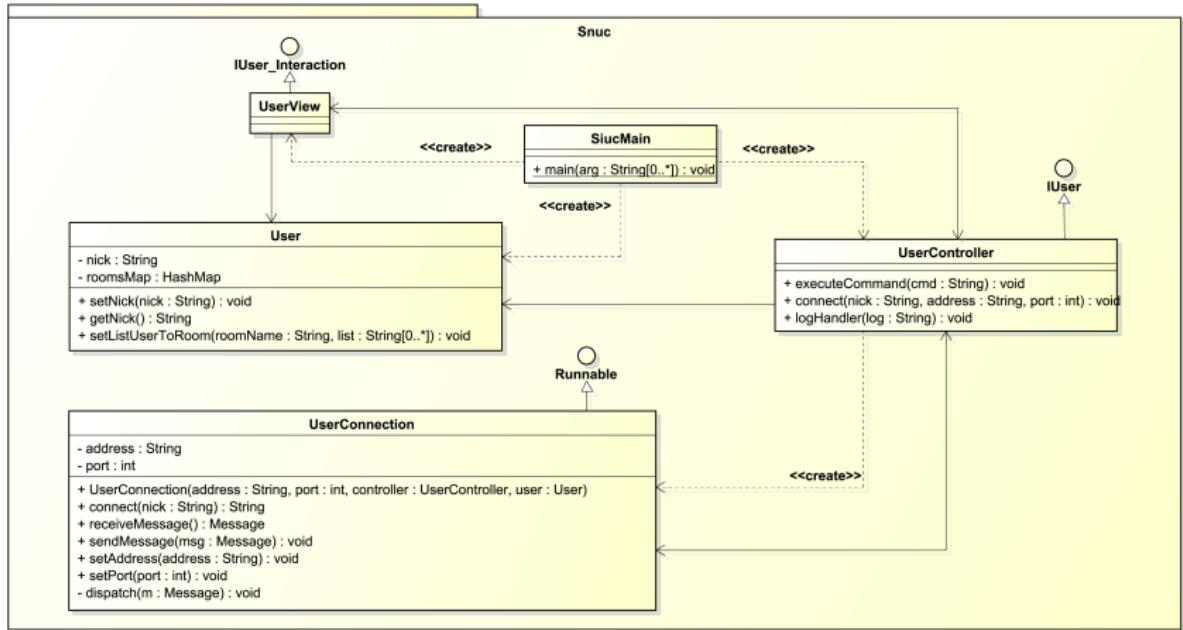
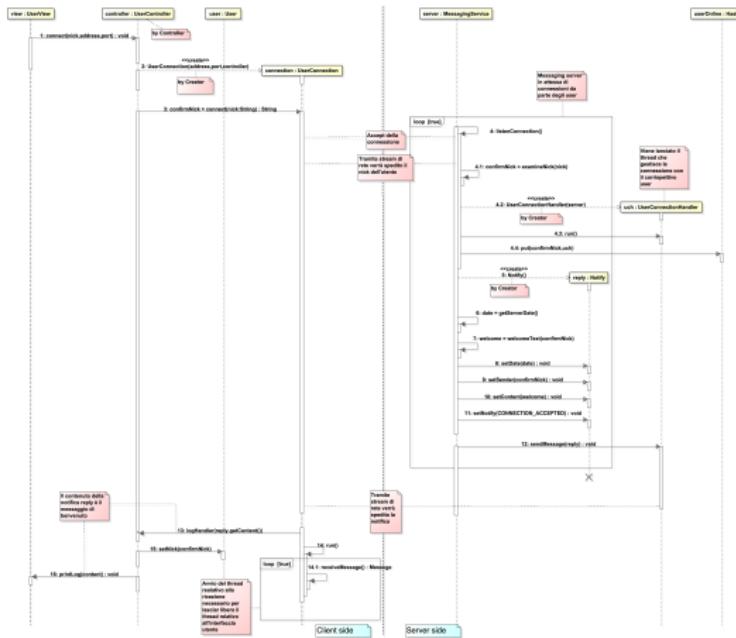


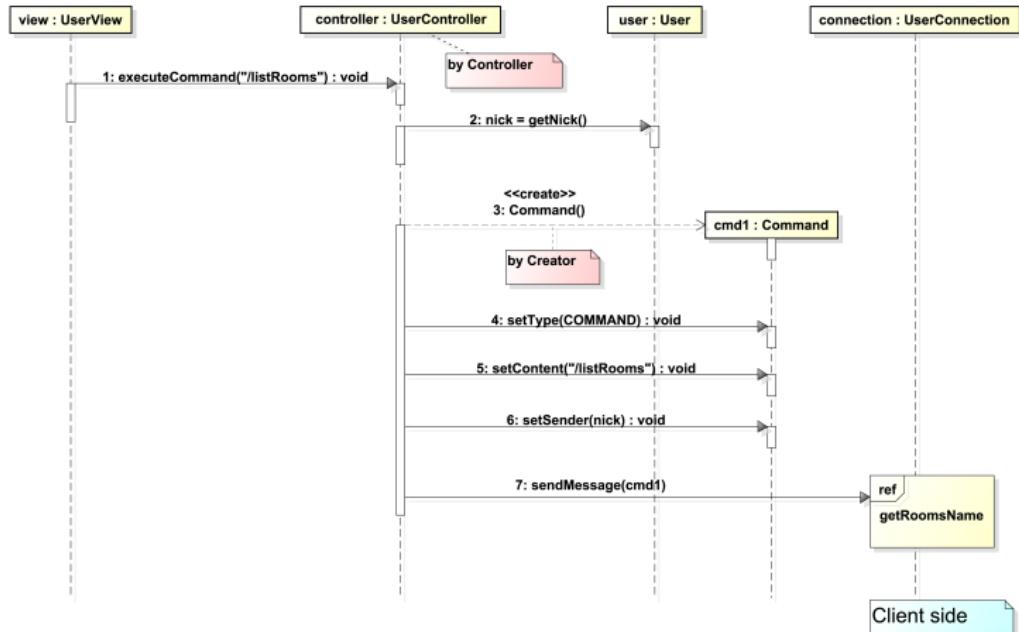
Figura 13 : DCD - Diagramma delle Classi: Package Snuc

## Iterazione 1: Progettazione, UC1\_RequestConnection



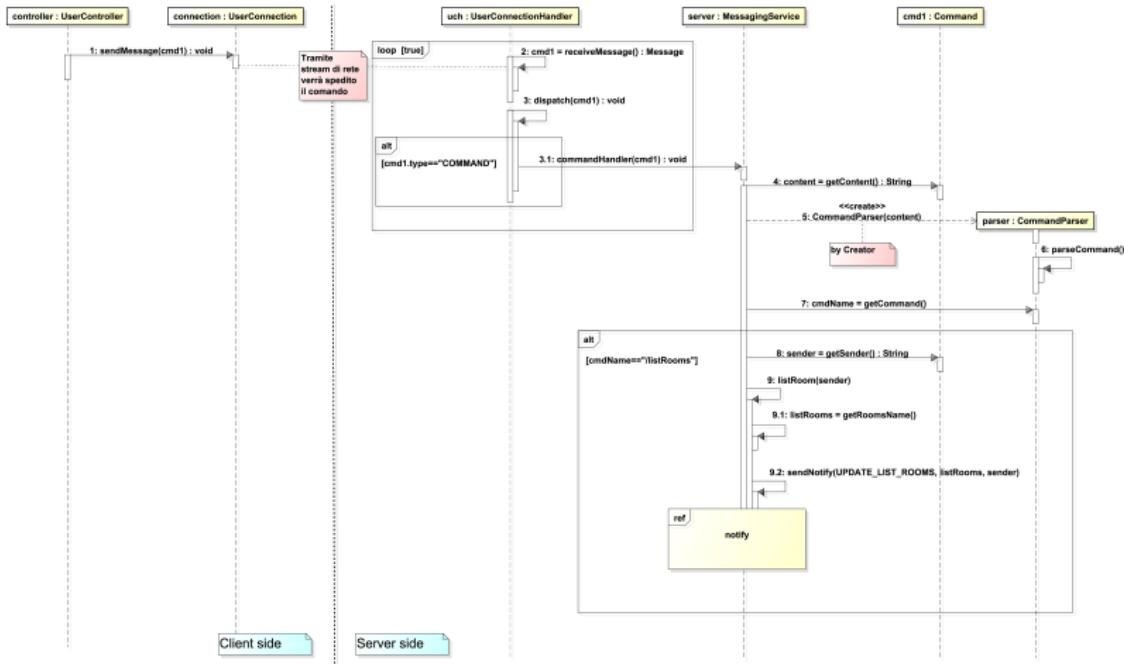
**Figura 14 : SSD - OP1-4: connect(nick), notify(response) del modello dominio (figura 7)**

## Iterazione 1: Progettazione, UC2\_AccessRoom - OP1



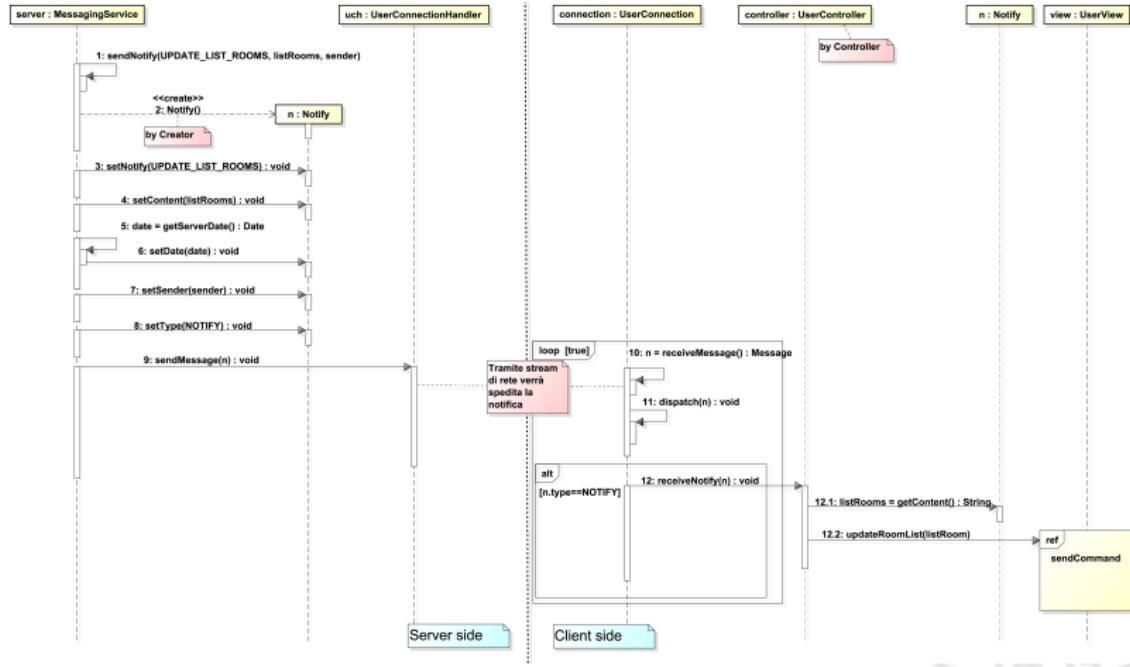
**Figura 15 :** SSD - OP1: `sendCommand(cmd1)` del modello di dominio (figura 10)

## Iterazione 1: Progettazione, UC2\_AccessRoom - OP2



**Figura 16 :** SSD - OP2: `getRoomsName()` del modello di dominio (figura 10)

## Iterazione 1: Progettazione, UC2\_AccessRoom - OP3



**Figura 17 :** SSD - OP3: `notify(list)` del modello di dominio (figura 10)

## Iterazione 1: Progettazione, UC2\_AccessRoom - OP4

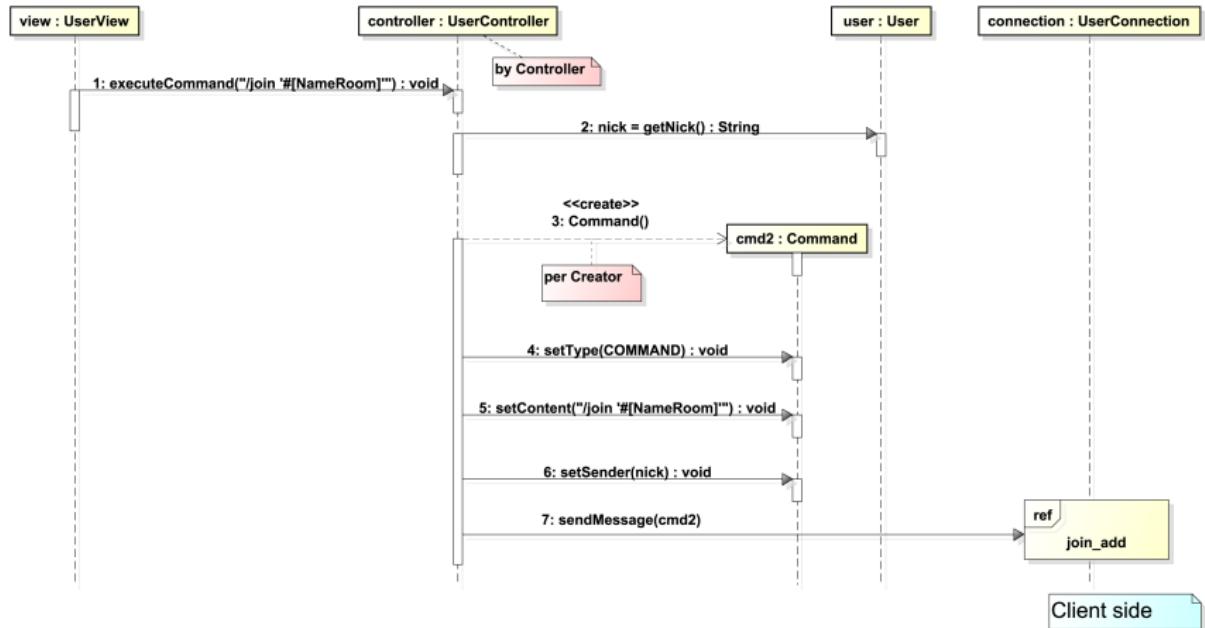


Figura 18 : SSD - OP4: sendCommand(cmd2) del modello di dominio (figura 10)

# Iterazione 1: Progettazione, UC2\_AccessRoom - OP5

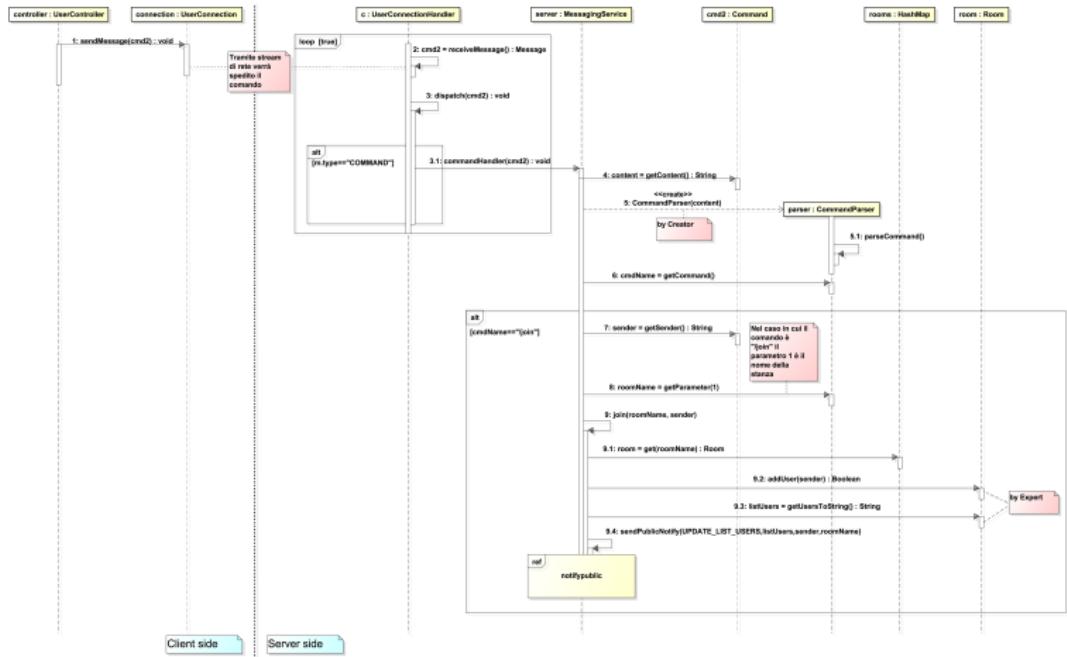


Figura 19 : SSD - OP5: joinRoom(sender,room), addUserToRoom() del modello di dominio (figura 10)

# Iterazione 1: Progettazione, UC2\_AccessRoom - OP6

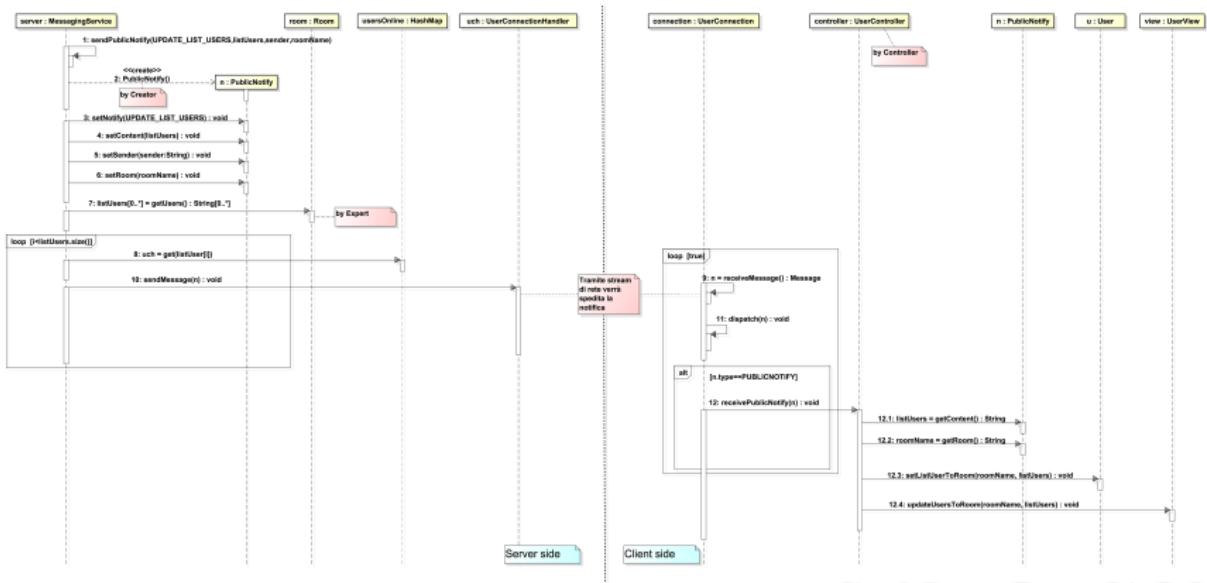


Figura 20 : SSD - OP6: `notifypublic(updateList)` del modello di dominio (figura 10)



## Descrizione: Refactoring Iterazione 1

Tra i requisiti iniziali del nostro progetto presenti nel documento di specifica dei requisiti, il cliente ha richiesto un software flessibile al funzionamento di diversi tipi di protocolli di comunicazione, quali ad esempio TCP e UDP. Con la struttura attuale del nostro progetto se volessimo soddisfare tale requisito sarebbe necessario apportare consistenti modifiche sia negli oggetti lato client che negli oggetti lato server. Per fare un esempio, attualmente il MessagingService si occupa di ascoltare le richieste di connessione mediante protocollo TCP. Se volessimo cambiare il tipo di comunicazione dovremmo cambiare radicalmente tale oggetto. Per eliminare questo tipo di dipendenze sarebbe opportuno adottare degli oggetti di supporto rendendo indipendenti gli oggetti del client e del server dal tipo di comunicazione. Abbiamo per questo pensato di adottare dei remote Proxy a cui possiamo dare delle responsabilità accessorie da connettore, senza cambiare gli aspetti funzionali del client e del server.

Abbiamo avuto la necessità di creare due tipi di remote Proxy:



## Descrizione: Refactoring Iterazione 1

ProxyMessagingService e ProxyUser. Il primo non è altro che una rappresentazione del MessagingService in locale al client; il secondo invece è una rappresentazione del User lato server. Il server avrà un ProxyUser per ogni client connesso. Sia il ProxyMessagingService che il MessagingService implementano la stessa interfaccia ovvero IMessagingService in modo tale che possiamo sostituire, modificare classi o componenti in futuro senza dover modificare il resto. Poiché il ProxyMessagingService non comunicherà direttamente con il MessagingService, abbiamo bisogno di un ulteriore intermediario che abbiamo chiamato Dispatcher. Questo oggetto non fa altro che ricevere le richieste codificate in maniera opportuna dal ProxyMessagingService e richiamare l'opportuno metodo del MessagingService.

Analogamente a quanto scritto sopra, il ProxyUser non comunicherà direttamente con l'UserController, ma anche in questo caso abbiamo bisogno di un intermediario, il Dispatcher. Sia il ProxyUser che il UserController implementano la stessa interfaccia IUser



## Descrizione: Refactoring Iterazione 1

in modo tale che, se necessario, possiamo sostituire, modificare classi o componenti in futuro senza dover modificare il resto. L'oggetto Dispatcher è stato progettato in modo da essere uguale sia per il client che per il server. Se viene settato come riferimento un IMessagingService il dispatcher sta lavorando lato server. Se invece viene settato come riferimento un IUser, allora il dispatcher sta lavorando lato client.

Rimane il problema di chi nel server si occupa di rimanere in ascolto per permettere l'instaurazione di connessioni. Abbiamo per questo pensato di introdurre un nuovo oggetto con tale funzionalità chiamato ConnectionHandler. Tale oggetto viene lanciato in un Thread a parte. Ad ogni richiesta di connessione da parte del client tale oggetto crea e lancia un Dispatcher e inoltre instanzia anche un ProxyUser che come abbiamo detto rappresenterà il nostro User in locale.

In base al tipo di protocollo utilizzato ogni oggetto di cui si è parlato precedentemente



## Descrizione: Refactoring Iterazione 1

avrà la sua implementazione. Per esempio, se volessimo utilizzare la classica comunicazione TCP, gli oggetti che implementeranno i relativi metodi delle classi astratte ProxyMessagingService, ProxyUser, Dispatcher e ConnectionHandler saranno:

- ProxyMessagingServiceTCP
- ProxyUserTCP
- DispatcherTCP
- ConnectionHandlerTCP

Se invece utilizziamo la comunicazione UDP:

- ProxyMessagingServiceUDP
- ProxyUserUDP
- DispatcherUDP
- ConnectionHandlerUDP





## Descrizione: Refactoring Iterazione 1

Relativamente al problema della creazione degli oggetti che si occupano della comunicazione abbiamo pensato di utilizzare un ulteriore pattern Gof: l'abstract factory. L'Abstract Factory è un pattern creazionale che ha lo scopo di fornire un'interfaccia per la creazione di famiglie di oggetti tra loro correlati o dipendenti limitando l'accoppiamento derivante dall'uso diretto delle classi concrete. Nel nostro caso le famiglie di oggetti sono relative al tipo di protocollo utilizzato. L'applicazione di questo pattern si rivela assai utile quando si vuole rendere un sistema indipendente dalle modalità di creazione, composizione e rappresentazione degli oggetti costituenti.

Abbiamo utilizzato la classe ConnectionFactory che è una classe astratta in cui sono definiti tutti i metodi che permetteranno la creazione degli oggetti relativi alla comunicazione. Tale classe implementa il metodo getConnectionFactory che, in base al file di configurazione, si occupa di instanziare le factory concrete (ConnectionFactoryTCP e ConnectionFactoryUDP) relative al protocollo utilizzato. Tali factory implementano i



## Descrizione: Refactoring Iterazione 1

metodi della classe astratta ConnectionFactory. Le famiglie di prodotti da noi considerate sono quelle relative ai protocolli TCP e UDP.

Per fornire un punto di accesso globale alle ConnectionFactory concrete abbiamo utilizzato il pattern Singleton per cui il metodo getConnectionFactory è un metodo statico e ritorna sempre la stessa istanza delle famiglia di factory caricata mediante file di configurazione.

Tutti gli oggetti che gestiscono la comunicazione tra client e server sono stati inseriti nel package Connector.

Un'ulteriore modifica apportata alla struttura del progetto è stata realizzata utilizzando il pattern Observer. Tale pattern comportamentale consente la definizione di associazioni di dipendenza di molti oggetti verso di uno, in modo che se quest'ultimo cambia il suo stato, tutti gli altri sono notificati e aggiornati automaticamente. Nel nostro caso gli oggetti che saranno osservati sono la Room e l'UserOnLine, classe che rappresenta



## Descrizione: Refactoring Iterazione 1

gli utenti connessi al server; mentre l'osservatore è il MessagingService.

Gli oggetti osservati, anche se ignari dell'identità del loro osservatore, non appena ci sarà un cambiamento del loro stato dovranno notificare l'osservatore il quale, in base al cambiamento notificato, dovrà richiamare l'opportuna procedura di gestione.

Ogni qual volta un utente si connette e viene aggiunto alla lista degli utenti online, il MessagingService verrà informato di tale cambiamento di stato e interviene inviando un messaggio di notifica di benvenuto che conferma che la connessione ha avuto successo.

Ogni qual volta viene registrato un nuovo utente ad una stanza, il MessagingService verrà informato di tale cambiamento di stato e interviene inviando una notifica pubblica a tutti gli utenti registrati nella stanza aggiornando la lista degli utenti presenti nella stanza.

## Refactoring Iterazione 1: Class Diagram Common UC1 e UC2

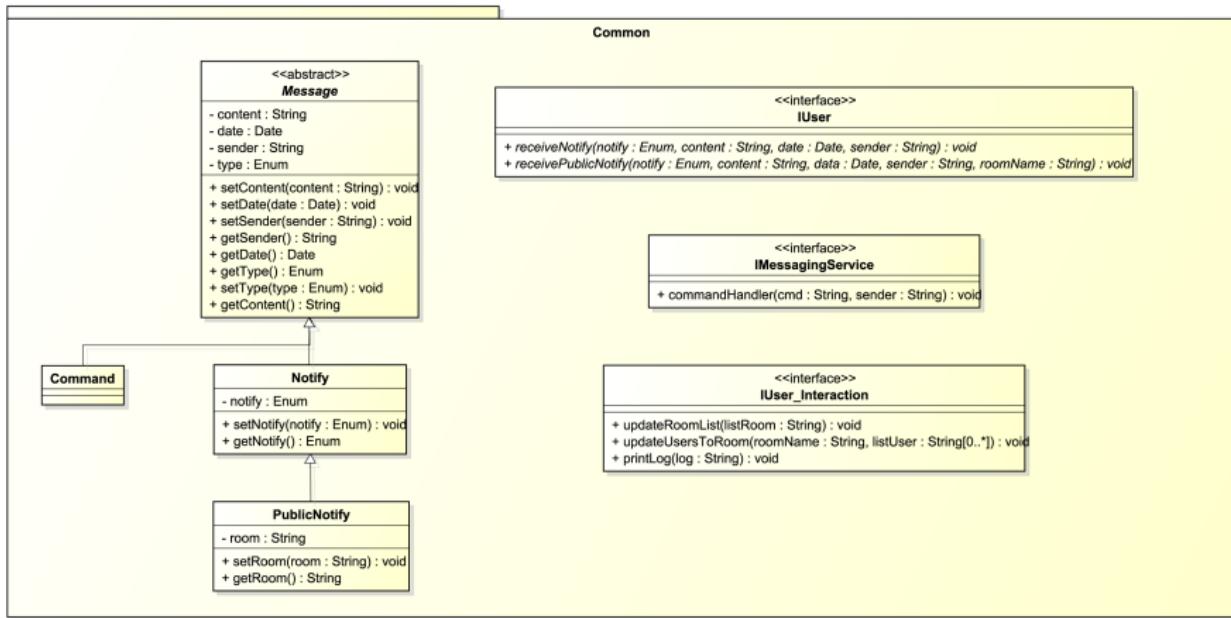


Figura 21 : DCD - Diagramma delle Classi: Package Common

## Refactoring Iterazione 1: Class Diagram Snuc UC1 e UC2

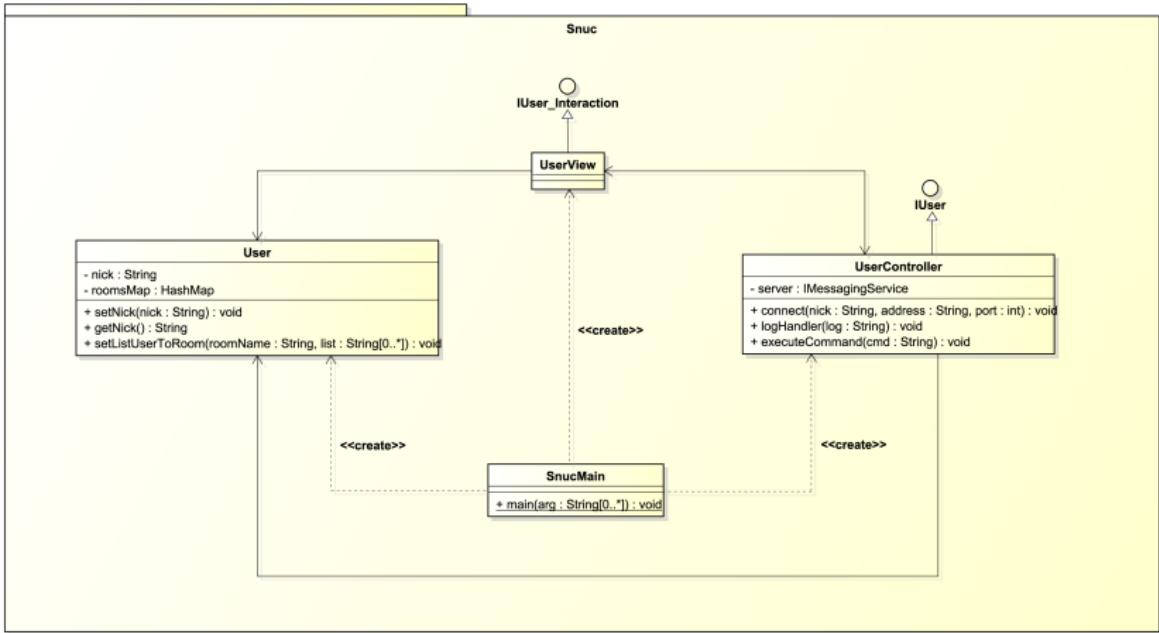


Figura 22 : DCD - Diagramma delle Classi: Package Snuc

## Refactoring Iterazione 1: Class Diagram Connector UC1 e UC2

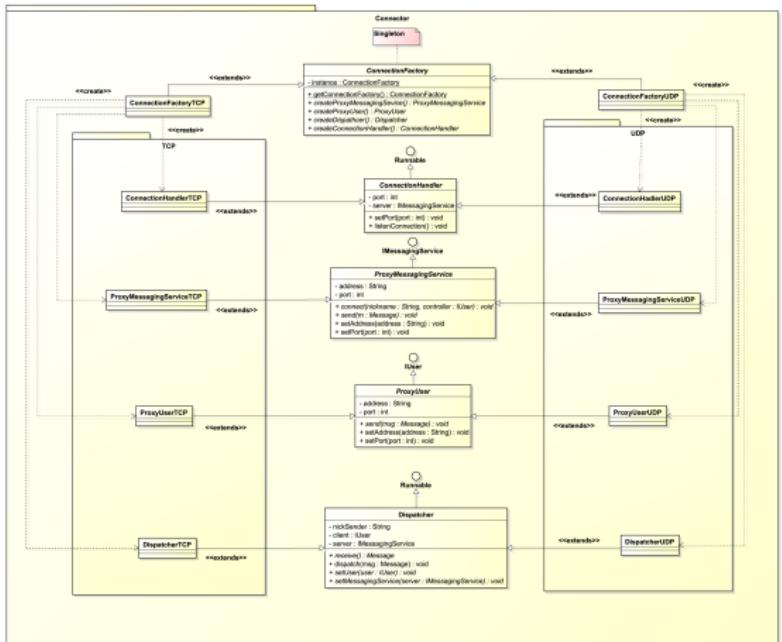


Figura 23 : DCD - Diagramma delle Classi: Package Connector

## Refactoring Iterazione 1: Class Diagram Snuc Server UC1 e UC2

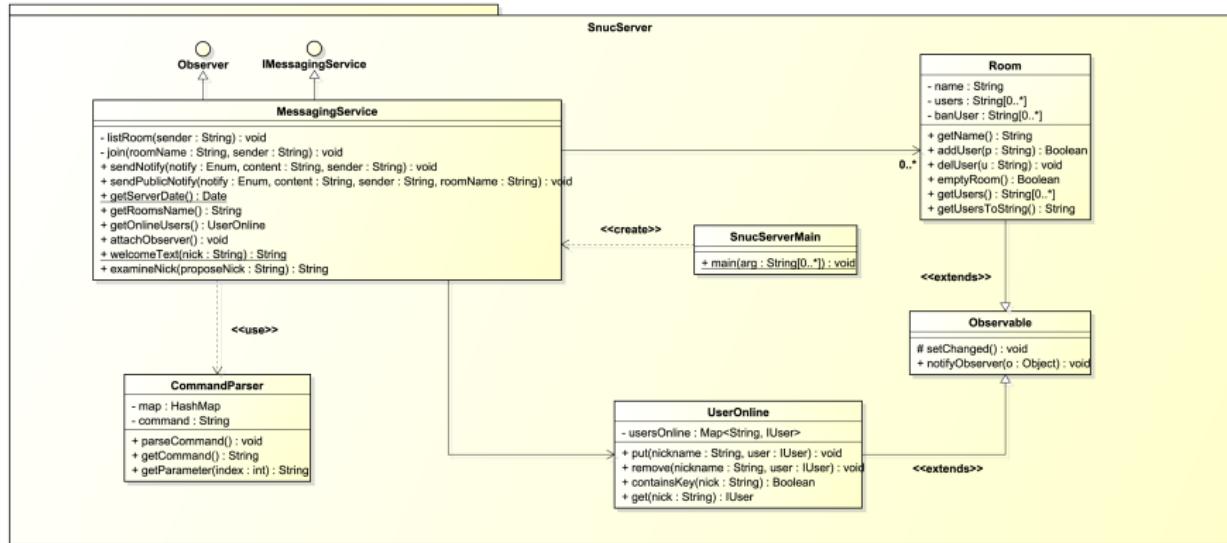


Figura 24 : DCD - Diagramma delle Classi: Package Snuc Server

## Refactoring Iterazione 1: UC1\_RequestConnection

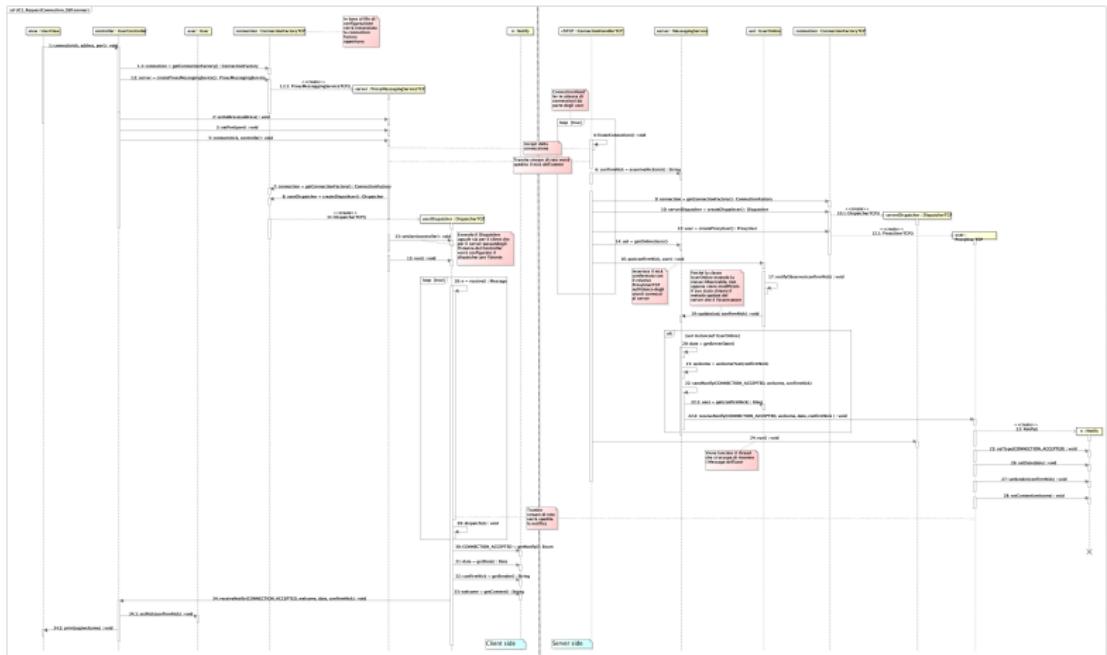


Figura 25 : SSD - OP1-4: connect(nick), notify(response) del modello dominio (figura 7)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP1

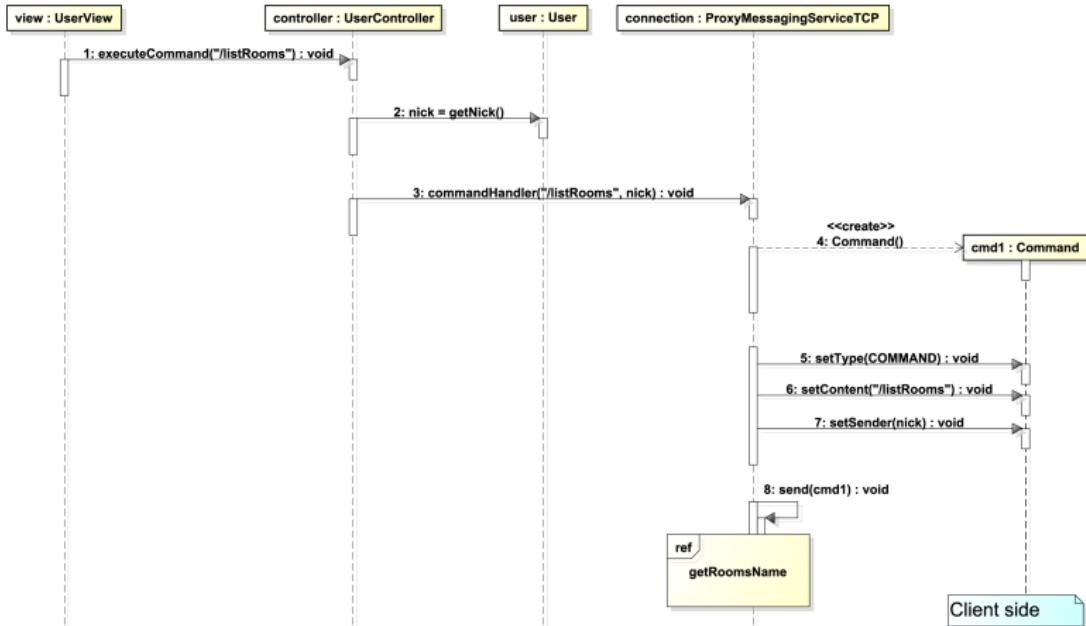


Figura 26 : SSD - OP1: sendCommand(cmd1) del modello di dominio (figura 10)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP2

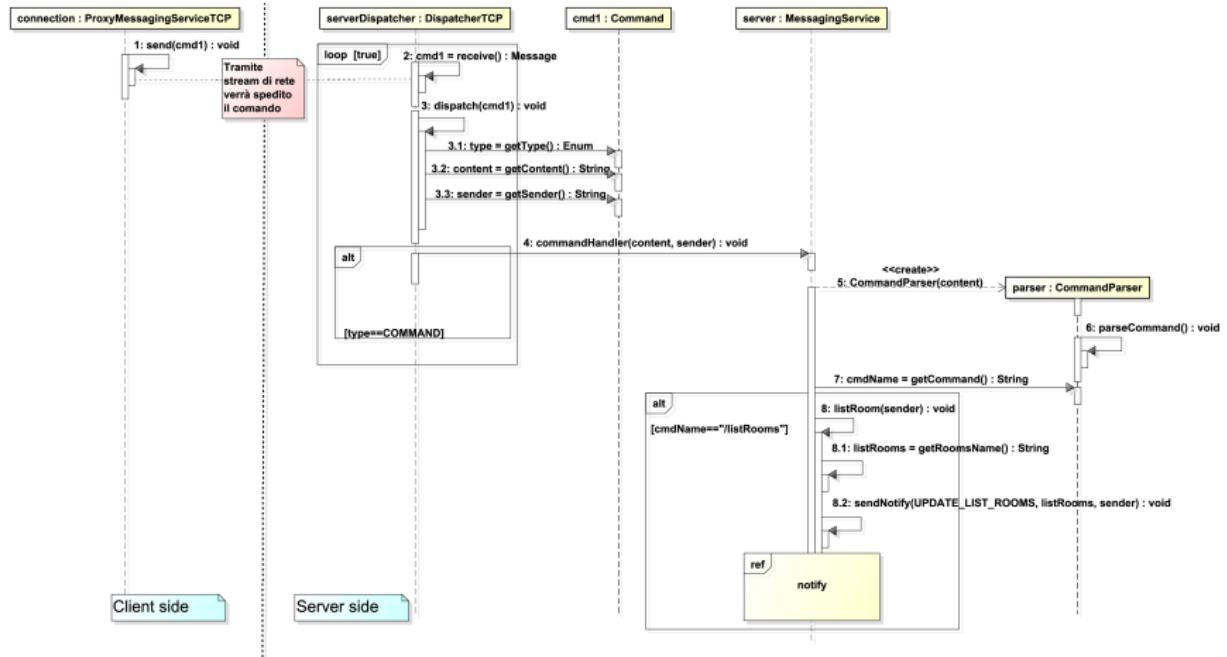


Figura 27 : SSD - OP2: getRoomsName() del modello di dominio (figura 10)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP3

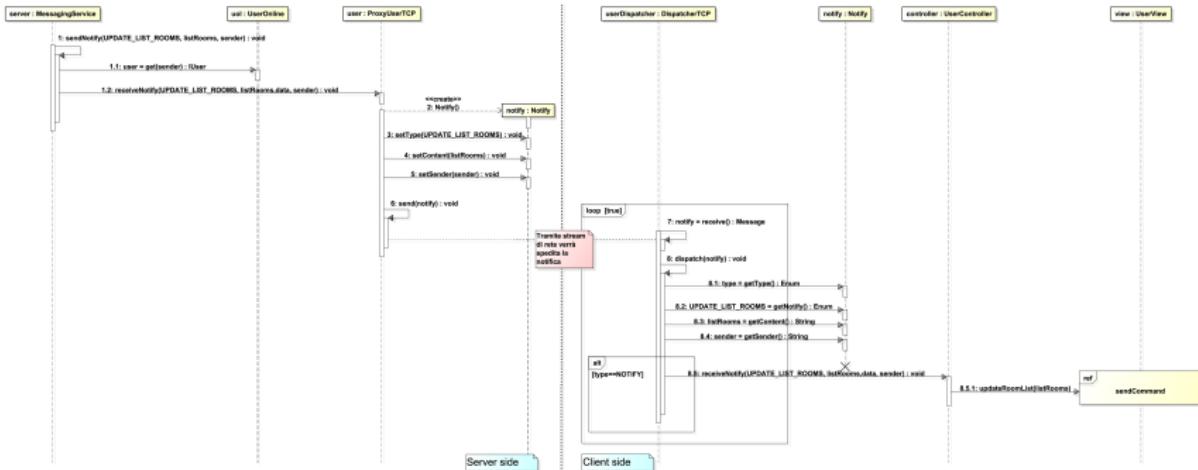
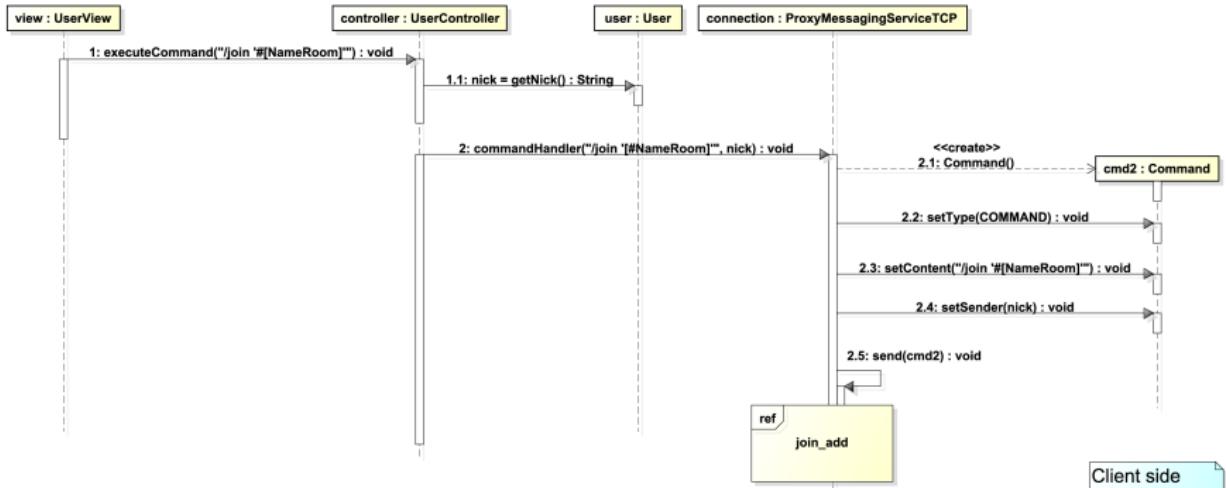


Figura 28 : SSD - OP3: notify(list) del modello di dominio (figura 10)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP4



**Figura 29 :** SSD - OP4: sendCommand(cmd2) del modello di dominio (figura 10)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP5

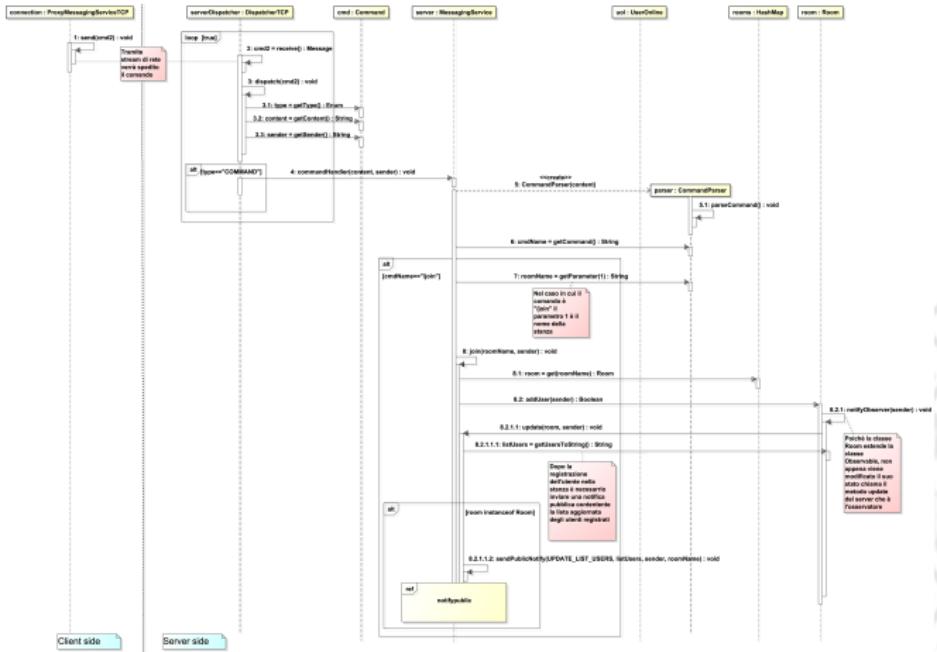


Figura 30 : SSD - OP5: `joinRoom(sender,room)`, `addUserToRoom()` del modello di dominio (figura 10)

## Refactoring Iterazione 1: UC2\_AccessRoom - OP6

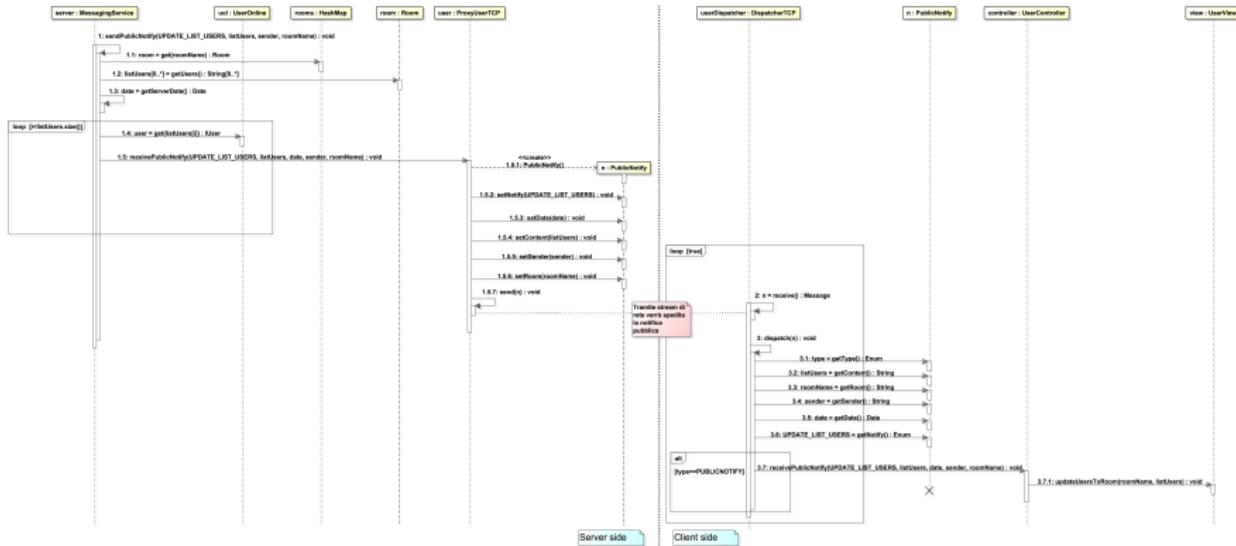


Figura 31 : SSD - OP6: `notifypublic(updateList)` del modello di dominio (figura 10)



# Refactoring Iterazione 1: Implementazione UC1 e UC2

*INSERIRE DESCRIZIONE*





## Refactoring Iterazione 1: Test UC1 e UC2

*INSERIRE ESEMPI DEI TEST EFFETTUATI*

**Listing 1 :** sample

```
1 public class HelloWorld {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hello, World");  
5     }  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20
```





## Refactoring Iterazione 1: Test UC1 e UC2

21  
22  
23  
24  
25  
26 }





## Descrizione Elaborazione - Iterazione 2

*INSERIRE DESCRIZIONE*





## Iterazione 2: Requisiti - UC3\_SendPublicMessage

**Tabella 7 :** Caso d'uso UC3\_SendPublicMessage

Nome caso d'uso	UC3_SendPublicMessage
Portata	Applicazione Smart Intelligent University Communications
Livello	Obiettivo Utente
Attore primario	SnucUser
Parti interessate e interessi	<ul style="list-style-type: none"> <li>▶ SnucUser, vuole che i messaggi siano inviati a ogni utente della stanza virtuale.</li> <li>▶ SnucAdmin, è interessato a supervisionare gli utenti del servizio affinché non ci siano abusi.</li> </ul>
Pre-condizioni	L'utente è registrato nella stanza in cui desidera inviare i messaggi.
Post-condizioni (garanzia di successo)	Ogni utente riceve il messaggio inviato.
Scenario principale di successo	<ol style="list-style-type: none"> <li>① L'utente inserisce in una opportuna area il messaggio da inviare.</li> <li>② Il messaggio viene inoltrato agli utenti presenti nella stanza selezionata.</li> </ol>



## Iterazione 2: Requisiti - UC3\_SendPublicMessage

Requisiti speciali (Requisiti Non Funzionali)	Comunicazione asincrona in cui lo scambio di informazioni avviene in tempo reale, senza sensibili pause tra invio e ricezione del messaggio
Elenco delle varianti tecnologiche	<ul style="list-style-type: none"> <li>▶ È possibile inviare messaggi confidenziali, autenticati e integri al server del servizio di messaggistica.</li> <li>▶ L'applicazione dovrebbe essere flessibile al funzionamento di diversi protocolli di comunicazione (es. TCP, UDP) e con diversi strati middleware (es. Socket, RMI)</li> </ul>
Frequenza di ripetizione	Potrebbe essere quasi ininterrotta
Varie e/o Problemi Aperti	//



## Descrizione Iterazione 2: Analisi - UC3\_SendPublicMessage

In questa iterazione, del caso d'uso UC3 è di interesse lo scenario principale di successo. Da esso è possibile identificare le seguenti classi concettuali:

- ▶ **User:** rappresenta il generico utente, caratterizzato da un “nickname”, connesso al servizio di messaggistica. Può richiedere la lista delle stanze, ricevere notifiche dal sistema centrale. Interagisce con il MessagingService richiedendo la registrazione e l'ingresso in una specifica stanza. *Può inviare messaggi pubblici agli utenti di una stanza.*



## Descrizione Iterazione 2: Analisi - UC3\_SendPublicMessage

- ▶ **MessagingService:** rappresenta ed incapsula il servizio di messaggistica nel suo complesso. Mantiene una lista di utenti connessi a tale sistema. Mantiene una lista di stanze e riceve tramite comandi richieste di ingresso da parte degli utenti. *Svolge il ruolo di smistatore di messaggi inviati dagli User.*
- ▶ **Message:** individua un generico messaggio scambiato tra utenti della chat o tra servizio di messaggistica e utente. È costituito da un “content” (contenuto del messaggio), da una “date” (rappresenta la data) e dal “sender” (mittente).



## Descrizione Iterazione 2: Analisi - UC3\_SendPublicMessage

- ▶ **Notify:** è una specializzazione del tipo Message ed è caratterizzata da un typeNotify che serve a distinguere il tipo di notifica (ad es. CONNECTIO\_ACCEPT nel caso in cui la connessione è stata stabilita correttamente, BAD\_COMMAND nel caso in cui il comando inviato dall'User non sia riconosciuto dal Server).
- ▶ **PublicNotify:** è una specializzazione di Notify e questo tipo di notifica viene ricevuta da tutti gli utenti registrati alla relativa stanza. Un esempio di PublicNotify è la notifica caratterizzata dal seguente typeNotify: UPDATE\_LIST\_USERS, grazie alla quale viene aggiornata la lista degli utenti registrati nella relativa stanza.



## Descrizione Iterazione 2: Analisi - UC3\_SendPublicMessage

- ▶ **Command:** è una specializzazione di Message e rappresenta il comando che viene inviato dall'User e ricevuto ed interpretato dal MessagingService (es. /join '#Medical' richiesta da parte dell'utente a registrarsi alla stanza Medical).
- ▶ **Room:** è caratterizzata da un nome. Ciascuna istanza individua una specifica stanza nella chat.
- ▶ **Register:** mantiene un riferimento all'insieme di partecipanti che in un certo istante sono presenti nella stanza.
- ▶ **PublicMessage:** è una specializzazione di Message ed individua un messaggio pubblico scambiato tra utenti della chat registrati nella stessa stanza.

## Iterazione 2: Analisi - UC3\_SendPublicMessage

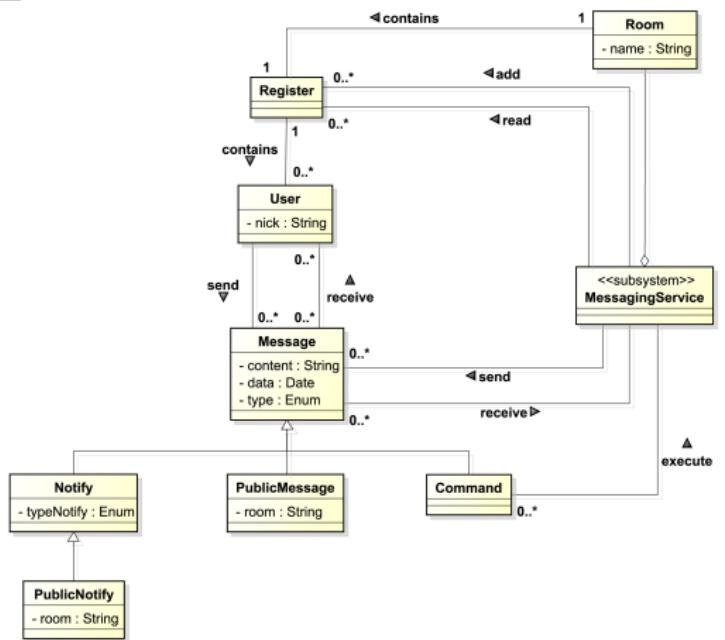


Figura 32 : UC3 - Modello di dominio

## Iterazione 2: Analisi - UC3\_SendPublicMessage

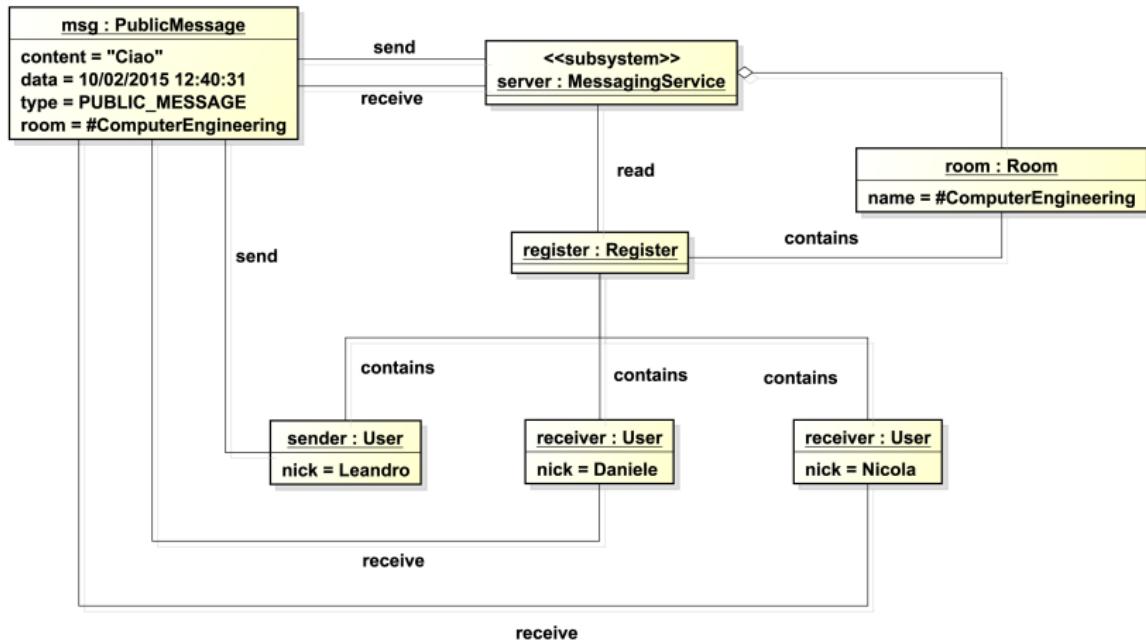


Figura 33 : UC3 - Oggetti di dominio

## Iterazione 2: Analisi - UC3\_SendPublicMessage

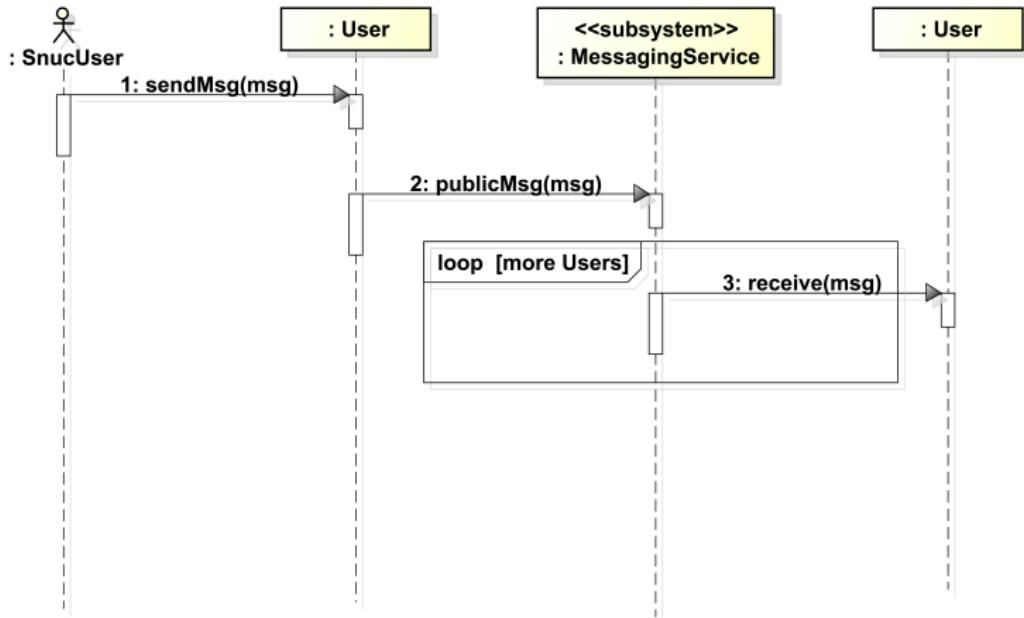


Figura 34 : UC3 - Diagramma di sequenza di sistema



## Iterazione 2: Analisi, UC3 contratto CO5

**Tabella 8 :** UC3 Contratto CO5 - publicMsg

Operazione	<b>publicMsg(msg:PublicMessage)</b>
Riferimenti	Caso d'uso: UC3_SendPublicMessage
Pre-condizione	L'utente è registrato nella stanza
Post-condizione	Gli utenti registrati nella stanza ricevono il messaggio inviato





## Descrizione Iterazione 2: Progettazione

In questa iterazione è stato sviluppato il caso d'uso relativo all'invio del messaggio pubblico. Rispetto al class diagram precedentemente mostrato è stata aggiunta la classe PublicMessage nel package Common. Tale classe, come la classe Notify e Command, è una sottoclasse di Message avente come attributo aggiuntivo il nome della stanza a cui è indirizzato il messaggio pubblico. Per la gestione dei messaggi pubblici sono state modificate le interfacce IUser, in cui è stato inserito un metodo per la ricezione dei messaggi pubblici, e IMessagingService nella quale invece è stato inserito un metodo per l'invio dei messaggi pubblici. Anche l'interfaccia IUserInteraction verrà modificata per permettere la visualizzazione nell'interfaccia utente del messaggio pubblico.

## Iterazione 2: Progettazione Class Diagram Common UC3

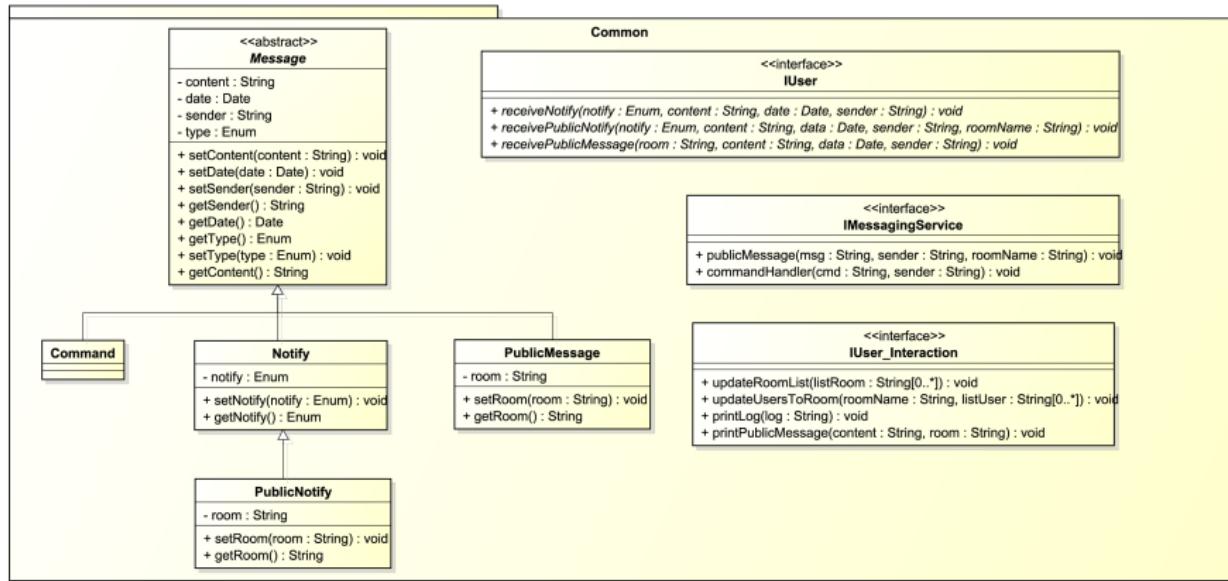


Figura 35 : DCD - Diagramma delle Classi: Package Common

## Iterazione 2: Progettazione - Class Diagram Snuc UC3

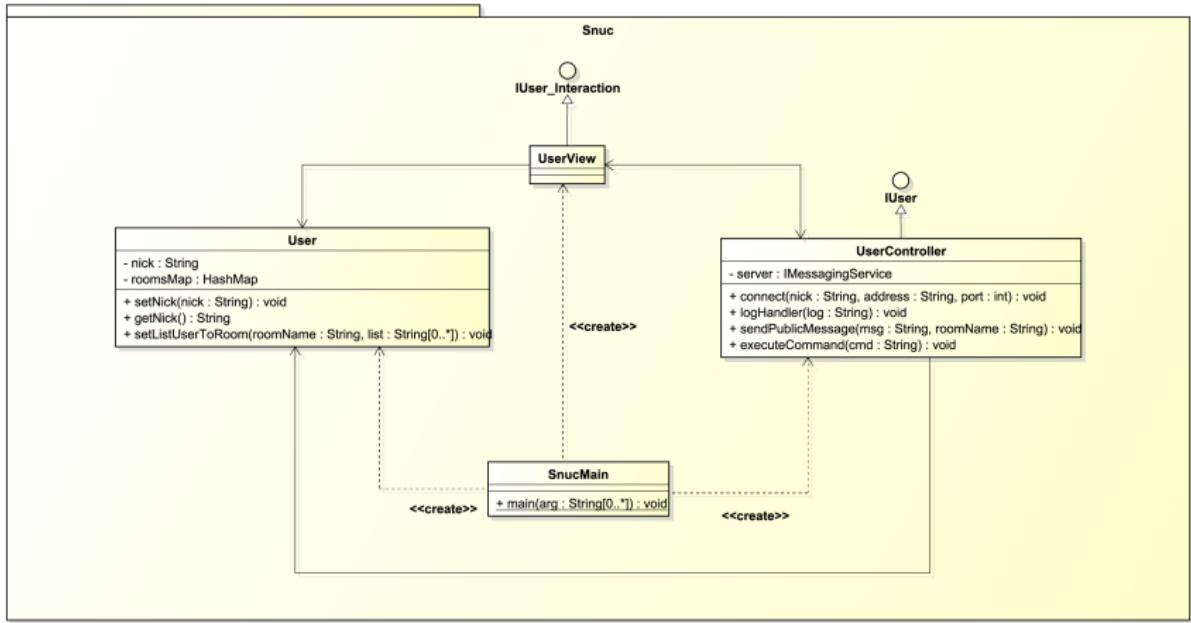
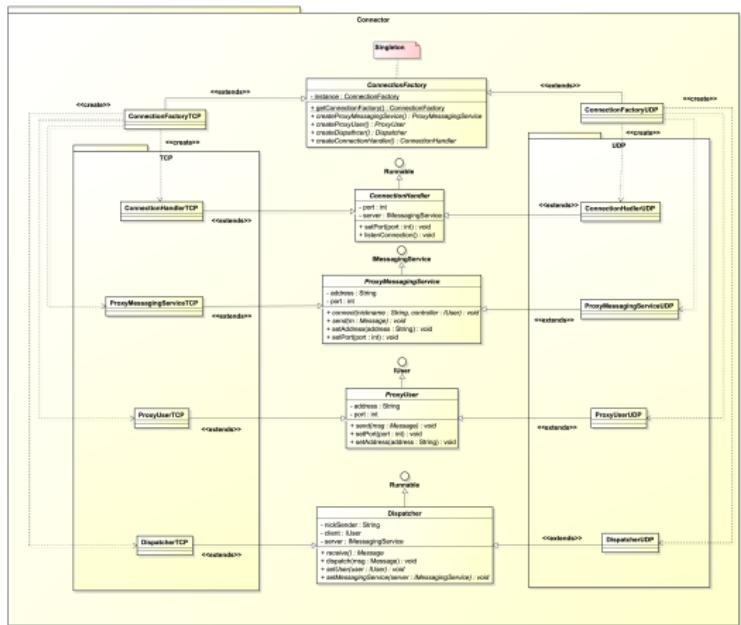


Figura 36 : DCD - Diagramma delle Classi: Package Snuc

## Iterazione 2: Progettazione - Class Diagram Connector UC3



**Figura 37 : DCD - Diagramma delle Classi: Package Connector**

## Iterazione 2: Progettazione - Class Diagram Snuc Server UC3

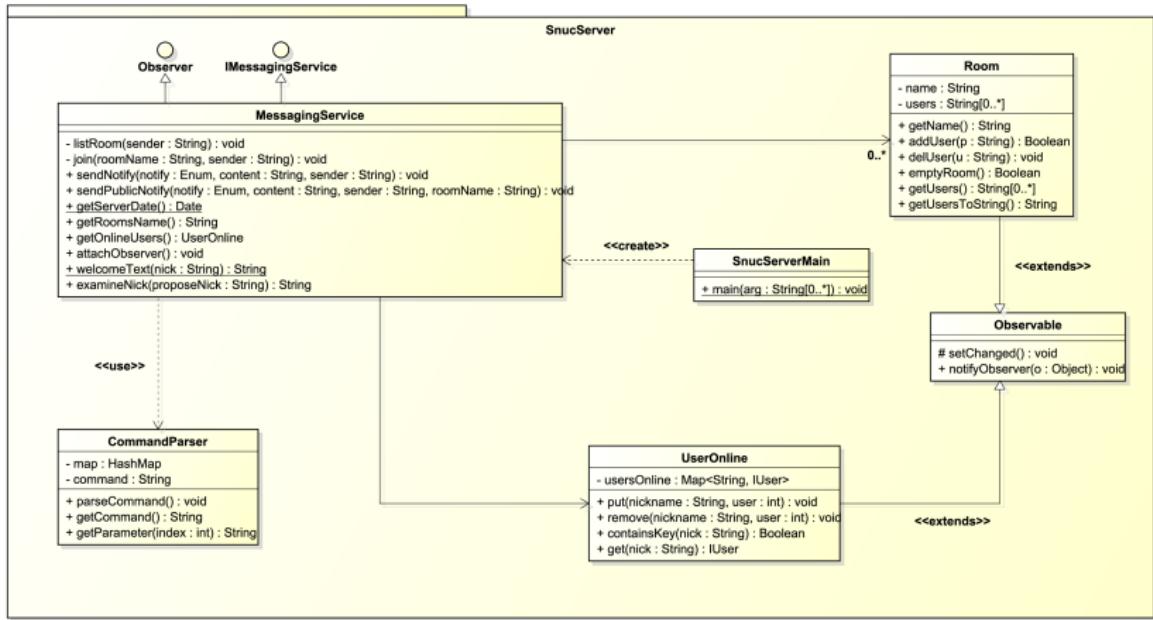
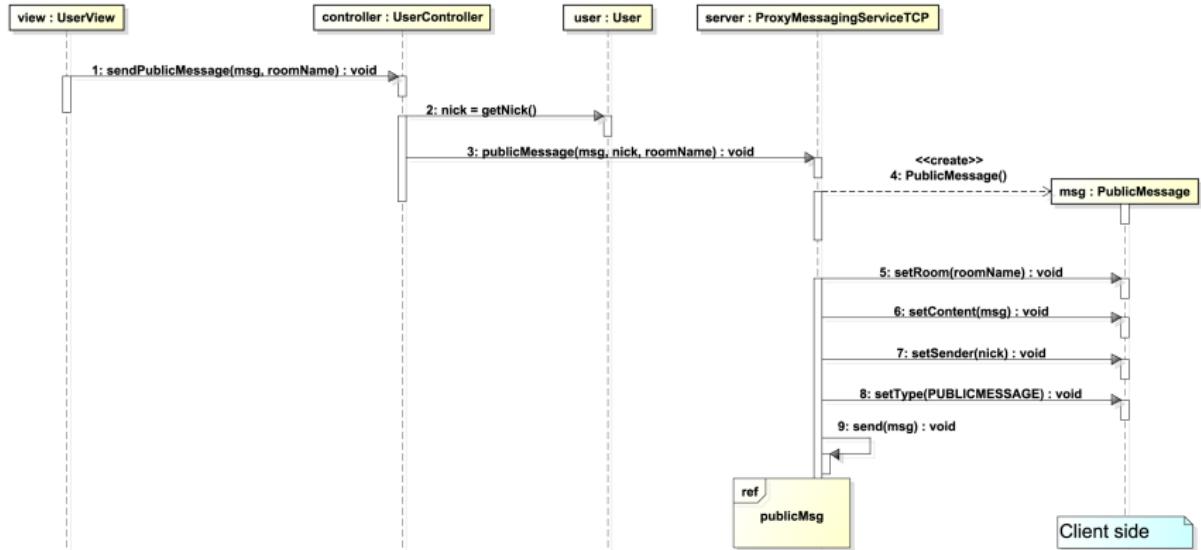


Figura 38 : DCD - Diagramma delle Classi: Package Snuc Server

## Iterazione 2: Progettazione, UC3\_SendPublicMessage - OP1



**Figura 39 :** SSD - OP1: sendMsg(msg) del modello di dominio (figura 34)

## Iterazione 2: Progettazione, UC3\_SendPublicMessage - OP2

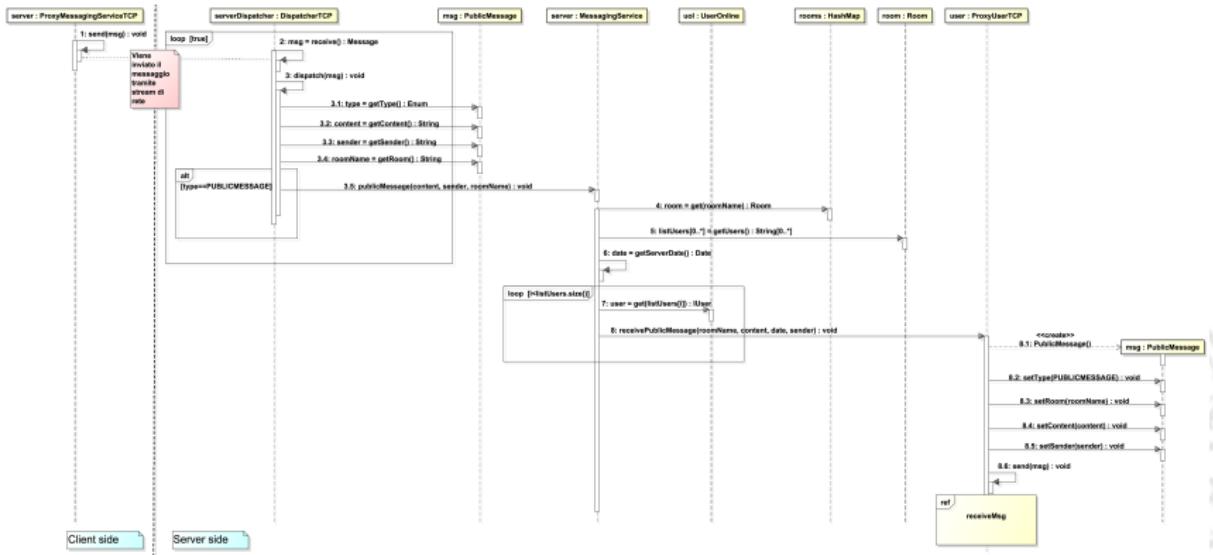
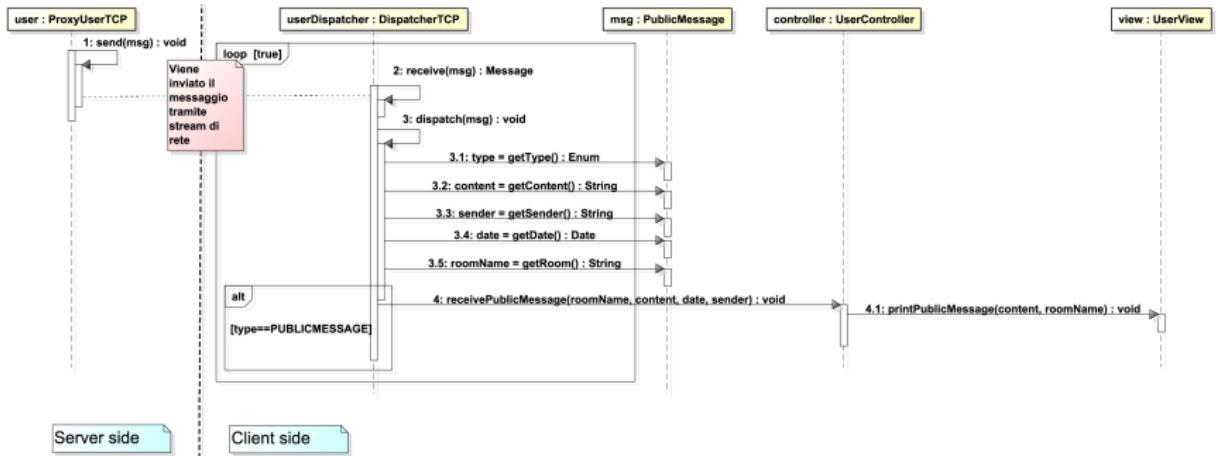


Figura 40 : SSD - OP2: pubblicMsg(msg) del modello di dominio (figura 34)

## Iterazione 2: Progettazione, UC3\_SendPublicMessage - OP3



**Figura 41 :** SSD - OP3: receive(msg) del modello di dominio (figura 34)





## Iterazione 2: Implementazione - UC3\_SendPublicMessage

*INSERIRE DESCRIZIONE*





## Descrizione Elaborazione: Iterazione 3

In questa terza iterazione è stato sviluppato il caso d'uso relativo all'invio del messaggio privato. Rispetto al class diagram precedentemente è stata aggiunta la classe `PrivateMessage` nel package `Common`. Tale classe, come la classe `Notify`, `Command` e `PublicMessage`, è una sottoclasse di `Message` avente come attributo aggiuntivo il nickname del destinatario a cui è indirizzato il messaggio privato.

Per la gestione dei messaggi privati sono state modificate le interfacce `IUser` in cui è stato inserito un metodo per la ricezione dei messaggi privati, e `IMessagingService` nella quale invece è stato inserito un metodo per l'invio dei messaggi privati. Anche l'interfaccia `IUserInteraction` verrà modificata per permettere la visualizzazione nell'interfaccia utente del messaggio privato.



## Iterazione 3: Requisiti - UC4\_SendPrivateMessage

**Tabella 9 :** Caso d'uso UC4\_SendPrivateMessage

Nome caso d'uso	UC3_SendRoomMessage
Portata	Applicazione Smart Intelligent University Communications
Livello	Obiettivo Utente
Attore primario	SnucUser
Parti interessate e interessi	SnucUser: vuole che i messaggi siano inviati all'utente selezionato presente nella stanza virtuale
Pre-condizioni	L'utente è registrato nella stanza in cui desidera inviare un messaggio ad un altro utente presente
Post-condizioni (garanzia di successo)	L'utente selezionato riceve il messaggio inviato
Scenario principale di successo	<ul style="list-style-type: none"><li>① L'utente seleziona il destinatario del messaggio privato.</li><li>② L'utente inserisce da tastiera il messaggio da inviare.</li><li>③ Il messaggio viene inoltrato al destinatario selezionato.</li></ul>



## Iterazione 3: Requisiti - UC4\_SendPrivateMessage

Requisiti speciali (Requisiti Non Funzionali)	Comunicazione asincrona in cui lo scambio di informazioni avviene in tempo reale, senza sensibili pause tra invio e ricezione del messaggio
Elenco delle varianti tecnologiche	<ul style="list-style-type: none"> <li>▶ È possibile inviare messaggi confidenziali, autenticati e integri al server del servizio di messaggistica.</li> <li>▶ L'applicazione dovrebbe essere flessibile al funzionamento di diversi protocolli di comunicazione (es. TCP, UDP) e con diversi strati middleware (es. Socket, RMI)</li> </ul>
Frequenza di ripetizione	Potrebbe essere quasi ininterrotta
Varie e/o Problemi Aperti	//



## Descrizione Iterazione 3: Analisi - UC4\_SendPrivateMessage

In questa iterazione, del caso d'uso UC4 è di interesse lo scenario principale di successo. Da esso è possibile identificare le seguenti classi concettuali:

- ▶ **User:** rappresenta il generico utente, caratterizzato da un “nickname”, connesso al servizio di messaggistica. Può richiedere la lista delle stanze, ricevere notifiche dal sistema centrale. Interagisce con il MessagingService richiedendo la registrazione e l'ingresso in una specifica stanza. Può inviare messaggi pubblici e *privati*.



## Descrizione Iterazione 3: Analisi - UC4\_SendPrivateMessage

- ▶ **MessagingService**: rappresenta ed incapsula il servizio di messaggistica nel suo complesso. Mantiene una lista di utenti connessi a tale sistema. Mantiene una lista di stanze e riceve tramite comandi richieste di ingresso da parte degli utenti. Svolge il ruolo di smistatore di messaggi inviati dagli User.
- ▶ **Message**: individua un generico messaggio scambiato tra utenti della chat o tra servizio di messaggistica e utente. È costituito da un “content” (contenuto del messaggio) , da una “date” (rappresenta la data) e dal “sender” (mittente).



## Descrizione Iterazione 3: Analisi - UC4\_SendPrivateMessage

- ▶ **Notify:** è una specializzazione del tipo Message ed è caratterizzata da un typeNotify che serve a distinguere il tipo di notifica (ad es. CONNECTIO\_ACCEPT nel caso in cui la connessione è stata stabilita correttamente, BAD\_COMMAND nel caso in cui il comando inviato dall'User non sia riconosciuto dal Server).
- ▶ **PublicNotify:** è una specializzazione di Notify e questo tipo di notifica viene ricevuta da tutti gli utenti registrati alla relativa stanza. Un esempio di PublicNotify è la notifica caratterizzata dal seguente typeNotify: UPDATE\_LIST\_USERS, grazie alla quale viene aggiornata la lista degli utenti registrati nella relativa stanza.



## Descrizione Iterazione 3: Analisi - UC4\_SendPrivateMessage

- ▶ **Command:** è una specializzazione di Message e rappresenta il comando che viene inviato dall'User e ricevuto ed interpretato dal MessagingService (es. /join '#Medical' richiesta da parte dell'utente a registrarsi alla stanza Medical).
- ▶ **Room:** è caratterizzata da un nome. Ciascuna istanza individua una specifica stanza nella chat.
- ▶ **Register:** mantiene un riferimento all'insieme di partecipanti che in un certo istante sono presenti nella stanza.
- ▶ **PublicMessage:** è una specializzazione di Message ed individua un messaggio pubblico scambiato tra utenti della chat registrati nella stessa stanza.

## Descrizione Iterazione 3: Analisi - UC4\_SendPrivateMessage

- ▶ **PrivateMessage:** *una specializzazione di Message ed individua un messaggio privato scambiato tra due utenti della chat registrati nella stessa stanza.*



## Iterazione 3: Analisi - UC4\_SendPrivateMessage

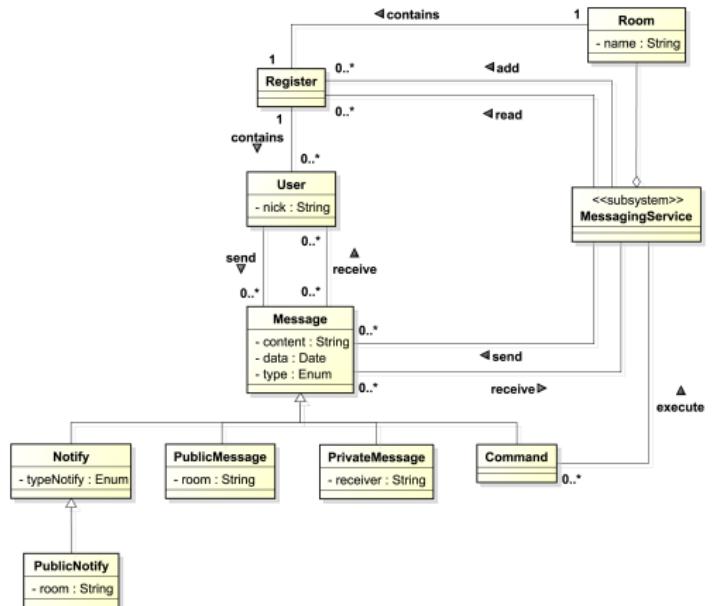


Figura 42 : UC3 - Modello di dominio

## Iterazione 3: Analisi - UC4\_SendPrivateMessage

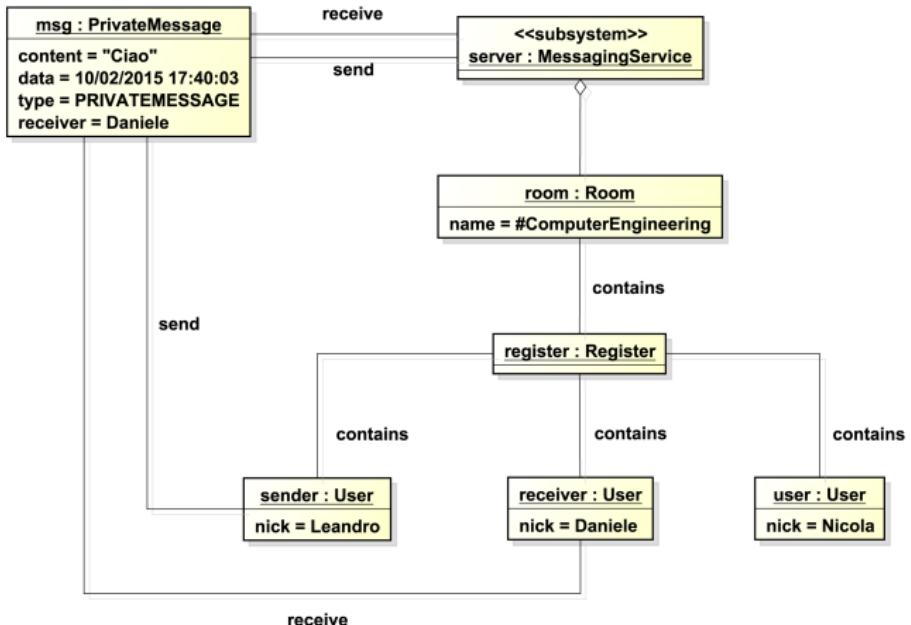


Figura 43 : UC3 - Oggetti di dominio

## Iterazione 3: Analisi - UC4\_SendPrivateMessage

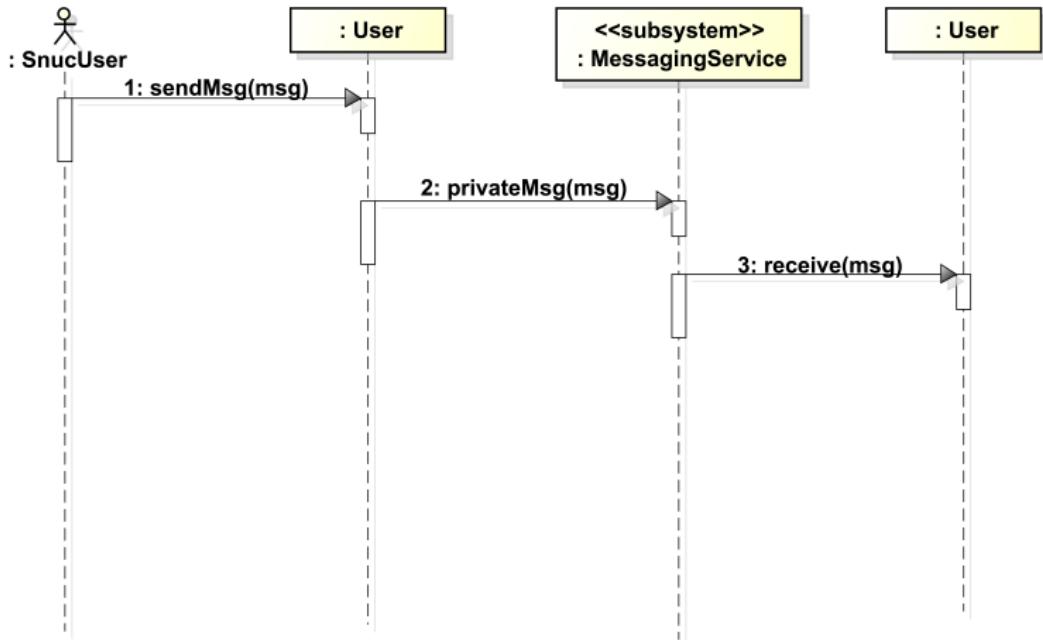


Figura 44 : UC4 - Diagramma di sequenza di sistema



## Iterazione 3: Analisi, UC4 contratto CO6

**Tabella 10 :** UC3 Contratto CO6 - privateMsg

Operazione	<i>privateMsg(msg:PublicMessage)</i>
Riferimenti	Caso d'uso: UC4_SendPrivateMessage
Pre-condizione	<ul style="list-style-type: none"><li>▶ L'utente è registrato nella stanza</li><li>▶ L'utente seleziona il destinatario del messaggio privato tra gli utenti registrati nella stanza</li></ul>
Post-condizione	Il destinatario riceve il messaggio privato



## Descrizione Iterazione 3: Progettazione

*INSERIRE DESCRIZIONE*





## Iterazione 3: Progettazione Class Diagram Common UC4

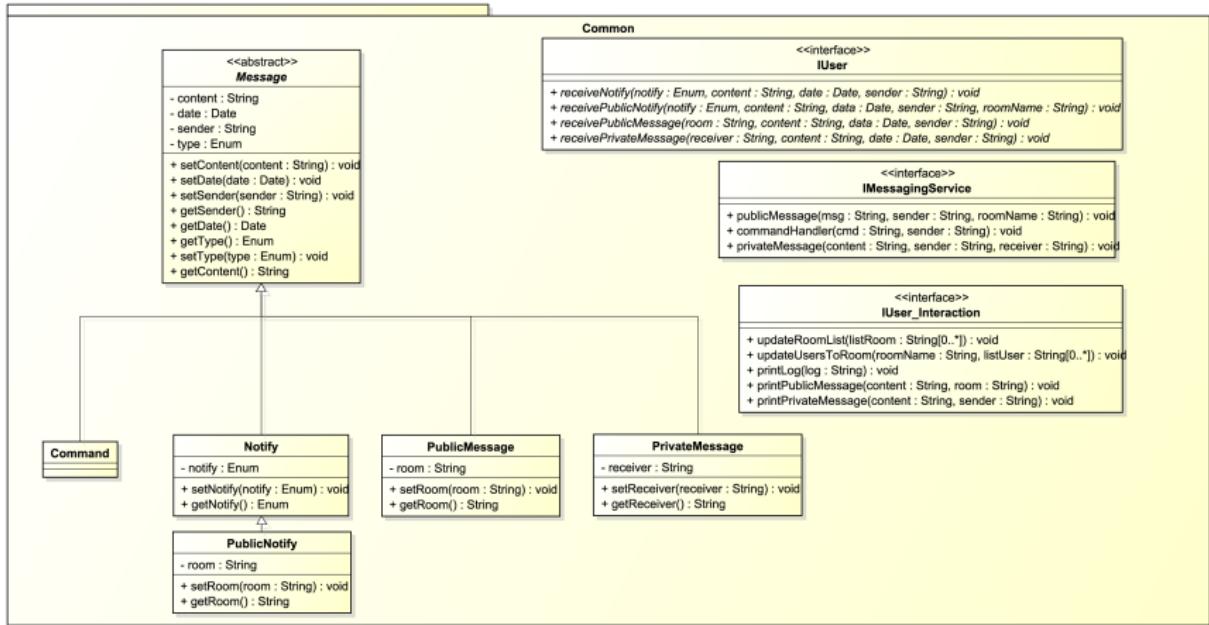


Figura 45 : DCD - Diagramma delle Classi: Package Common

## Iterazione 3: Progettazione Class Diagram Snuc UC4

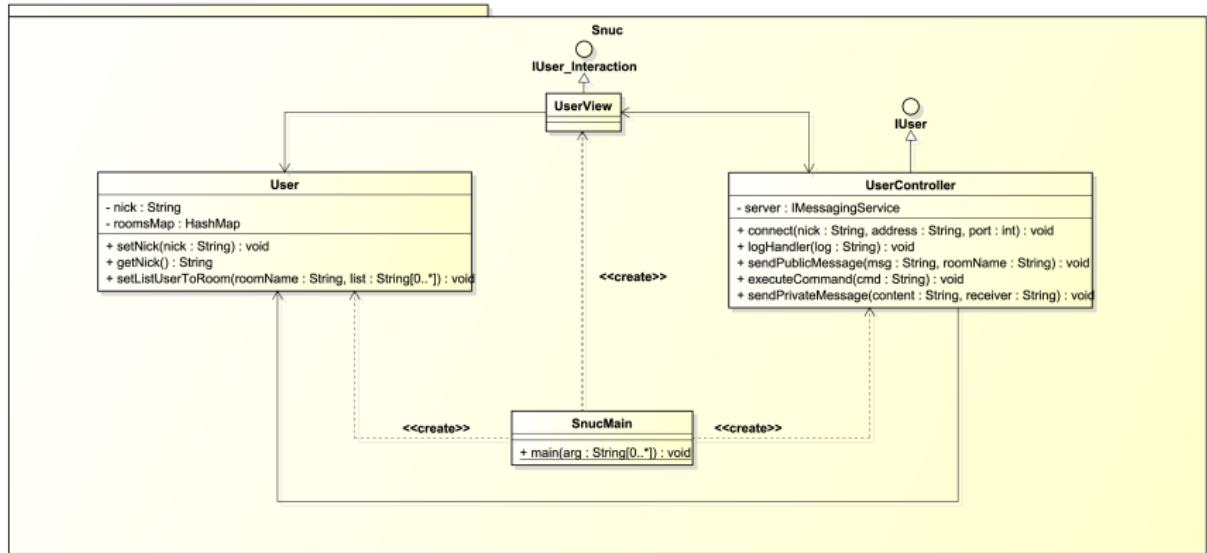


Figura 46 : DCD - Diagramma delle Classi: Package Snuc

## Iterazione 3: Progettazione Class Diagram Connector UC4

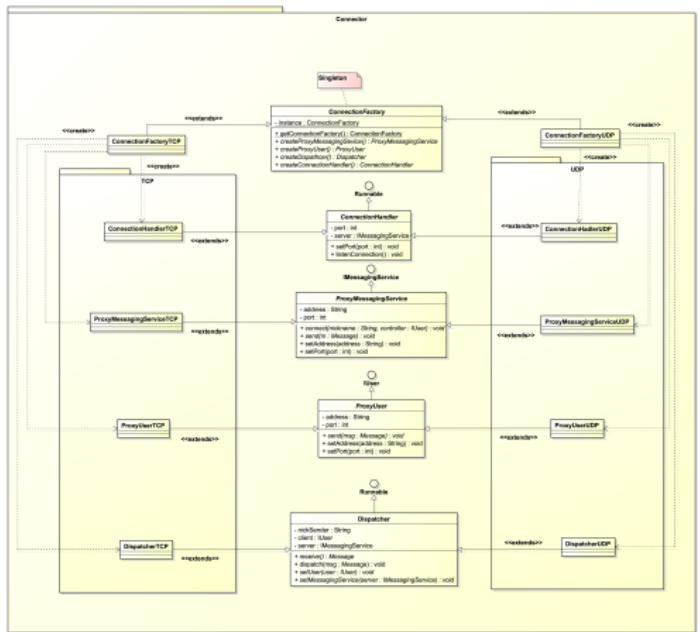


Figura 47 : DCD - Diagramma delle Classi: Package Connector

## Iterazione 3: Progettazione Class Diagram Snuc Server UC4

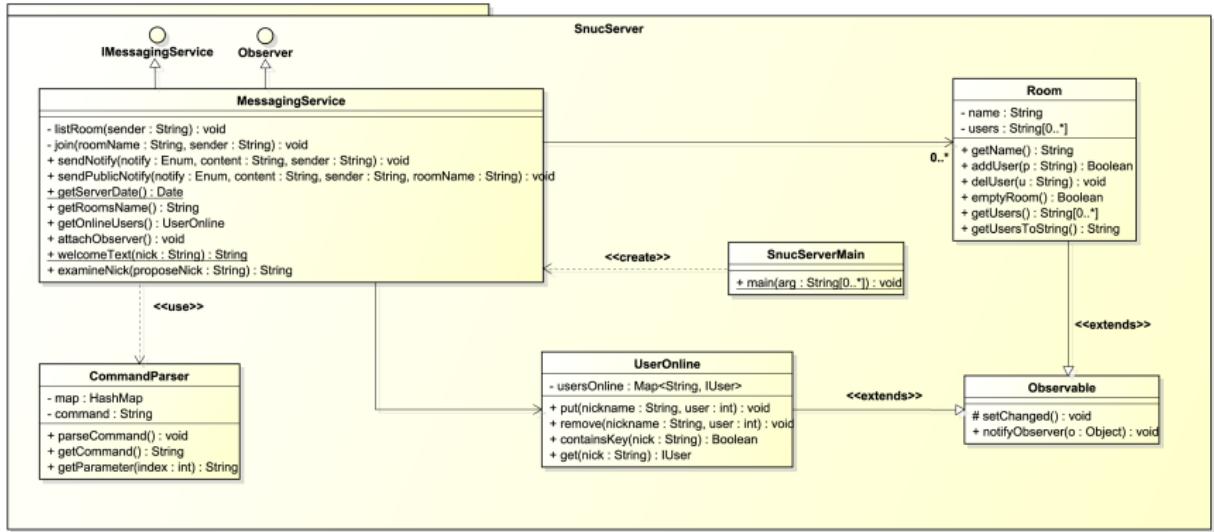


Figura 48 : DCD - Diagramma delle Classi: Package Snuc Server

## Iterazione 3: Progettazione, UC4\_SendPrivateMessage - OP1

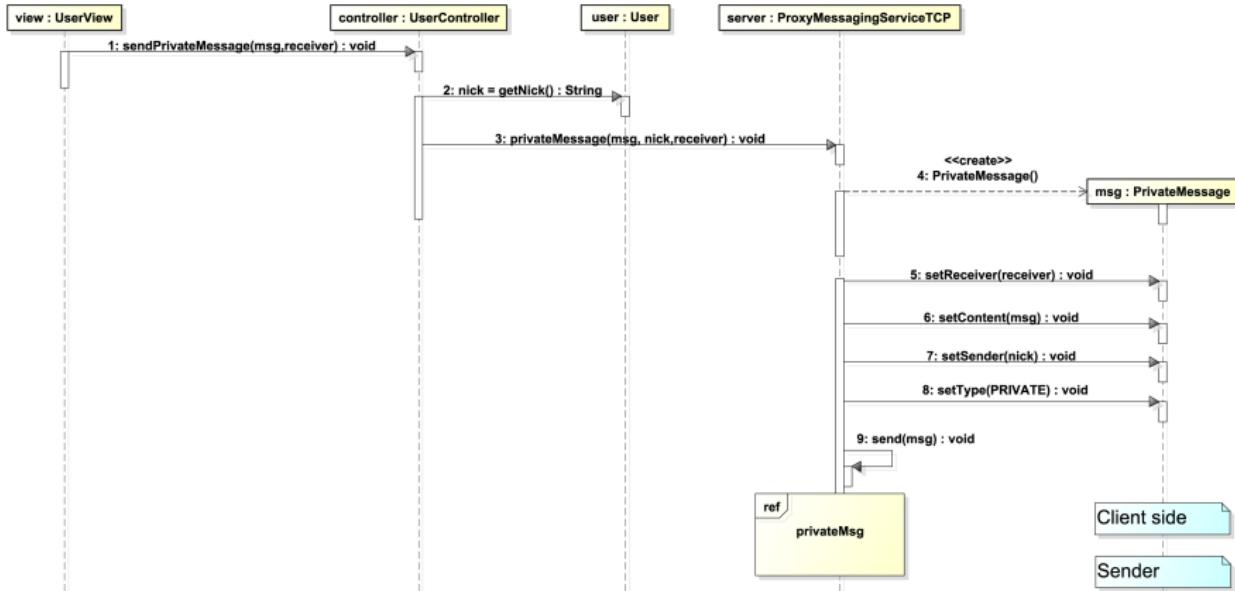


Figura 49 : SSD - OP1: sendMsg(msg) del modello di dominio (figura 44)

## Iterazione 3: Progettazione, UC4\_SendPrivateMessage - OP2

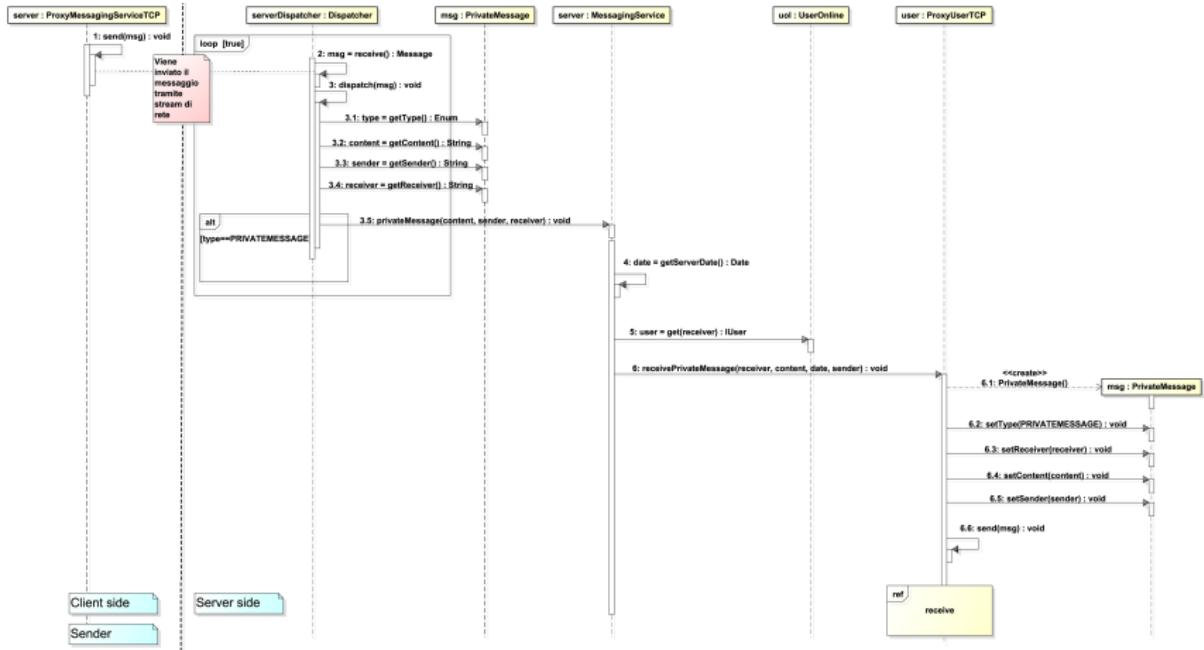
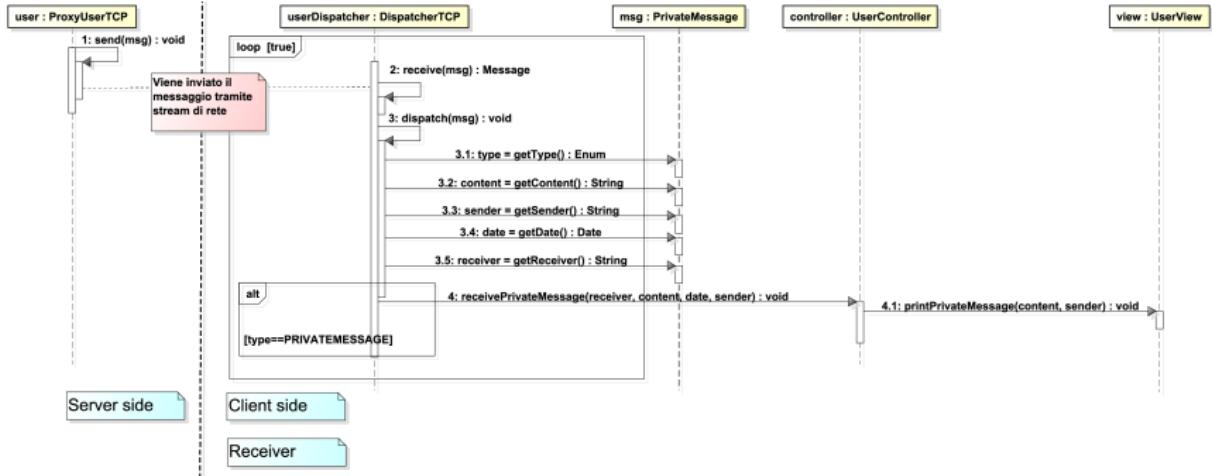


Figura 50 : SSD - OP2: privateMsg(msg) del modello di dominio (figura 44)

## Iterazione 3: Progettazione, UC4\_SendPrivateMessage - OP3



**Figura 51 :** SSD - OP3: receive(msg) del modello di dominio (figura 44)



## Iterazione 3: Implementazione - UC4\_SendPrivateMessage

*INSERIRE DESCRIZIONE*





## Bigliografia e Sitografia



**Herbert Schildt**

*Java SE 7. La guida completa.*  
McGraw-Hill, 2012



**Martin Fowler, L. Baresi, S. Gaburri**

*UML distilled. Guida rapida al linguaggio di modellazione standard*  
Pearson Education Italia, 2010



**Craig Larman, Luca Cabibbo**

*Applicare UML e i pattern: analisi e progettazione orientata agli oggetti*  
Pearson Education Italia, 2005



**M.L. Liu**

*Distributed Computing: Principles and Applications*  
Addison-Wesley, 2003



**Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides**

*Design patterns - Elementi per il riuso di software ad oggetti*  
Pearson Education Italia, 1995



**Pattern GOF - Giuseppe Dell'Abate's Blog**

Riferimenti: <https://dellabate.wordpress.com/category/gof-pattern/>



**Test unità con Junit4**

Riferimenti: <http://junit.org>





## Bigliografia e Sitografia



### The Java Tutorials

Riferimenti: <http://docs.oracle.com/javase/tutorial/>



### Java Platform, Standard Edition 8 API Specification

Riferimenti: <http://docs.oracle.com/javase/8/docs/api/>



### Git –everything-is-local

Riferimenti: <http://git-scm.com/>



### git - la guida tascabile

Riferimenti: <http://rogerdudler.github.io/git-guide/index.it.html>



### Astah.net: UML and Modeling Tools

Riferimenti: <http://astah.net/>



### NetBeans Documentation

Riferimenti: <https://netbeans.org/kb/docs/java/quickstart.html>

