



Facoltà di Ingegneria - Università di Catania

Smart Network University Communications

Software Engineering Projects

Docente

Prof. Orazio Tomarchio

Studenti

Russo Leandro
Invincibile Daniele
Didomenico Nicola

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

09 Marzo 2015



Indice

1. Indice

1.1. Prefazione

2. Ideazione - Iterazione 1

- 2.1. Iterazione 1: Requisiti - Documento di Visione
- 2.2. Iterazione 1: Requisiti - Diagrammi dei casi d'uso
- 2.3. Iterazione 1: Requisiti - Glossario
- 2.4. Iterazione 1: Requisiti - UC1_RequestConnection
- 2.5. Iterazione 1: Requisiti - UC2_AccessRoom
- 2.6. Iterazione 1: Analisi - UC1_RequestConnection
- 2.7. Iterazione 1: Analisi - UC2_AccessRoom
- 2.8. Iterazione 1: Progettazione - Class Diagram UC1 e UC2
- 2.9. Iterazione 1: Progettazione - SSD UC1_RequestConnection
- 2.10. Iterazione 1: Progettazione - SSD UC1_AccessRoom



Indice

3. Refactoring Iterazione 1

- 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
- 3.5. 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 - Descrizione
- 4.4. Iterazione 2: Progettazione Class Diagram UC3
- 4.5. Iterazione 2: Progettazione - SSD UC3_SendPublicMessage
- 4.6. Iterazione 2: Implementazione - UC3_SendPublicMessage



Indice

5. Elaborazione - Iterazione 3

- 5.1. Iterazione 3: Requisiti - UC4_SendPrivateMessage
- 5.2. Iterazione 3: Analisi - UC4_SendPrivateMessage
- 5.3. Iterazione 3: Progettazione - Descrizione
- 5.4. Iterazione 3: Progettazione Class Diagram UC4
- 5.5. Iterazione 3: Progettazione - SSD UC4_SendPrivateMessage
- 5.6. Iterazione 3: Implementazione - UC4_SendPrivateMessage

6. Conclusioni

- 6.1. Conclusioni
- 6.2. Software Utilizzati
- 6.3. Repository GitHub Progetto
- 6.4. 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: Ideazione (Inception), Elaborazione (Elaboration), Costruzione (Construction) e Transizione (Transition), come si può vedere nella figura 2.

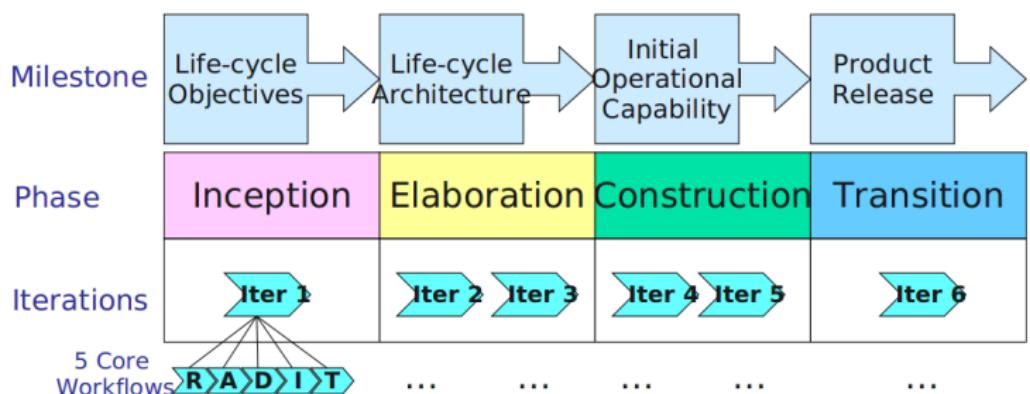


Figura 2 : Fasi dell'Unified Process



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.

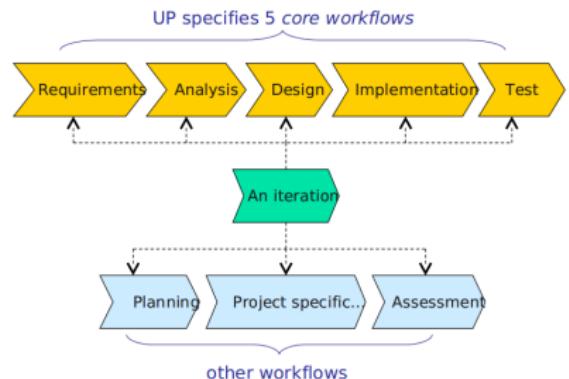


Figura 3 : Iterazioni dell'Unfiled Process



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 Descrizione dei casi d'uso in formato breve e dettagliato;
- ⑤ Prototipo UI (User Interface);
- ⑥ Modelli di dominio del sistema dei vari casi d'uso in formato dettagliato;
- ⑦ Diagrammi - Oggetti di dominio del sistema dei vari casi d'uso in formato dettagliato;
- ⑧ Diagrammi - Sequenza di dominio dei vari casi d'uso in formato dettagliato ;



Documenti del progetto realizzati

- ⑨ Contratti delle operazioni;
- ⑩ System Sequence Diagram (SSD) e Domain Class Diagrams (DCD) di progetto del sistema dei vari casi d'uso in formato dettagliato;
- ⑪ Codice sorgente dei diagrammi UML realizzati con Astah;
- ⑫ 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;
- ⑬ Test del software realizzati tramite il framework JUnit 4.0 di Java;
- ⑭ Documentazione tramite la Javadocs del codice sorgente;
- ⑮ Guida utente dell'avvio e configurazione del software realizzato;



Documenti del progetto realizzati

- ⑯ Documenti di installazione, downloading dei repository di GitHub;
- ⑰ Documentazione realizzata in \LaTeX .





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.



Descrizione realtà d'interesse

In tale sistema si richiedono le seguenti specifiche:

- 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.



Descrizione realtà d'interesse

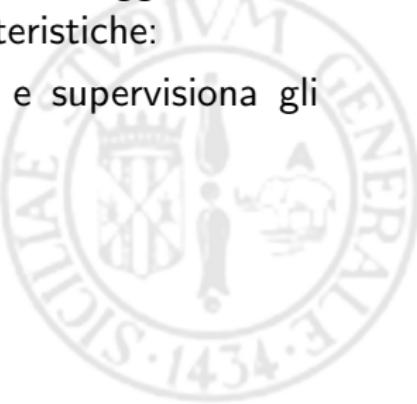
- In ogni istante il sistema prevede la presenza di una tipologia di utente particolare, chiamato amministratore.
I compiti di un amministratore sono i seguenti:
 - ▶ può creare o eliminare stanze del servizio di messaggistica;
 - ▶ inviare messaggi di avviso ad un utente;
 - ▶ in caso di comportamenti irregolari è possibile espellere un utente dalla stanza. L'utente espluso non può più rientrare fin quando questo non sarà rimosso dalla lista dei partecipanti bannati della stanza.
- ▶ 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. Gli utilizzatori del sistema avranno 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 ad 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

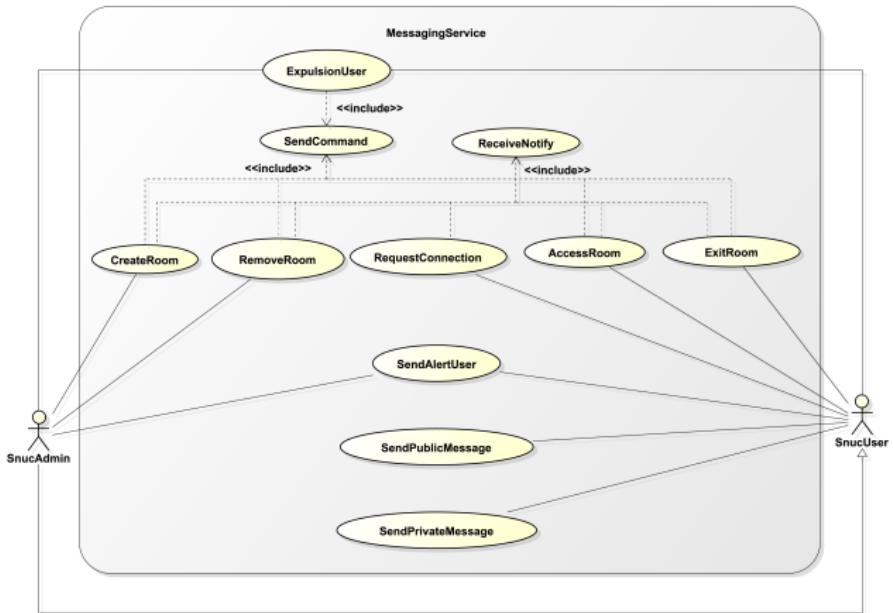


Figura 4 : Diagrammi dei casi uso



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 **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.



Iterazione 1: Requisiti - Modello dei casi d'uso

- ⑥ **CreateRoom.** L'amministratore del servizio di messaggistica è responsabile della creazione delle stanze che potranno successivamente essere visitate dai partecipanti.
- ⑦ **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 patecipanti presenti nella stanza, i quali potranno in seguito richiedere l'ingresso in una nuova stanza del servizio.
- ⑧ **SendAlertUser.** L'amministratore può in ogni momento inviare un messaggio di avviso ad uno specifico partecipante al servizio.



Iterazione 1: Requisiti - Modello dei casi d'uso

- ⑨ **ExpulsionUser.** L'amministratore può in ogni momento espellere da una stanza uno specifico partecipante.
Descrizione breve dei sottocasi d'uso identificati:
- ⑩ **ReceiveNotify.** Il sistema manda dei messaggi di notifica, che permettono di aggiornare l'utente di un cambiamento dello stato del sistema.
- ⑪ **SendCommand.** Il sistema è in grado di ricevere e interpretare dei comandi.





Iterazione 1: Requisiti - 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: rappresenta ed incapsula il servizio di messaggistica nel suo complesso.



Iterazione 1: Requisiti - 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: particolare messaggi di avviso inviato 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 Network 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"> ① L'utente inserisce un nickname, l'address e la porta del server. ② Il sistema esamina il nickname inviato dall'utente e verifica se è presente una omonimia. ③ Il sistema conferma l'inserimento tra gli utenti online inviando una notifica di benvenuto.



Iterazione 1: Requisiti - UC1_RequestConnection

Estensioni (o flussi alternativi)	<p>1A - SnucUser inserisce parametri errati:</p> <ul style="list-style-type: none"> ▶ Viene visualizzato un messaggio di errore e viene richiesto nuovamente l'inserimento di tali parametri. <p>2A - Omonimia del nickname:</p> <ul style="list-style-type: none"> ▶ Il sistema cambia il nickname aggiungendo il carattere “_” al nickname (es. _nickname). ▶ Il sistema conferma l'inserimento tra gli utenti online inviando una notifica di benvenuto.
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 Network 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"> ① L'utente richiede una lista di stanze presenti nel sistema di messaggistica. ② Il sistema invia la lista delle stanze. ③ L'utente seleziona la stanza tra quelle presenti in lista. ④ Il sistema registra l'utente alla stanza. ⑤ Il sistema visualizza a video gli utenti presenti nella stanza. ⑥ Il sistema visualizza un'area pubblica dove vengono mostrati tutte le conversazioni in corso dal quel momento in poi.
Estensioni (o flussi alternativi)	3A - Il SnucUser inserisce una stanza non in elenco: <ul style="list-style-type: none"> ① Il sistema invia un messaggio di errore.
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	//



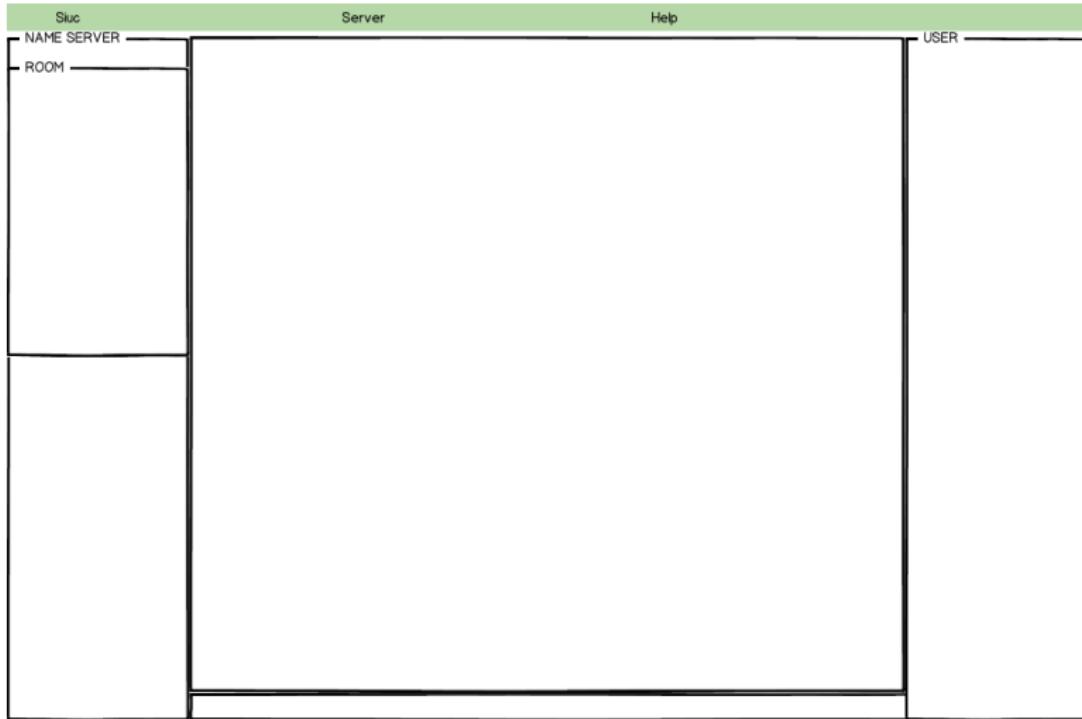
Descrizione Prototipo User Interface

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.



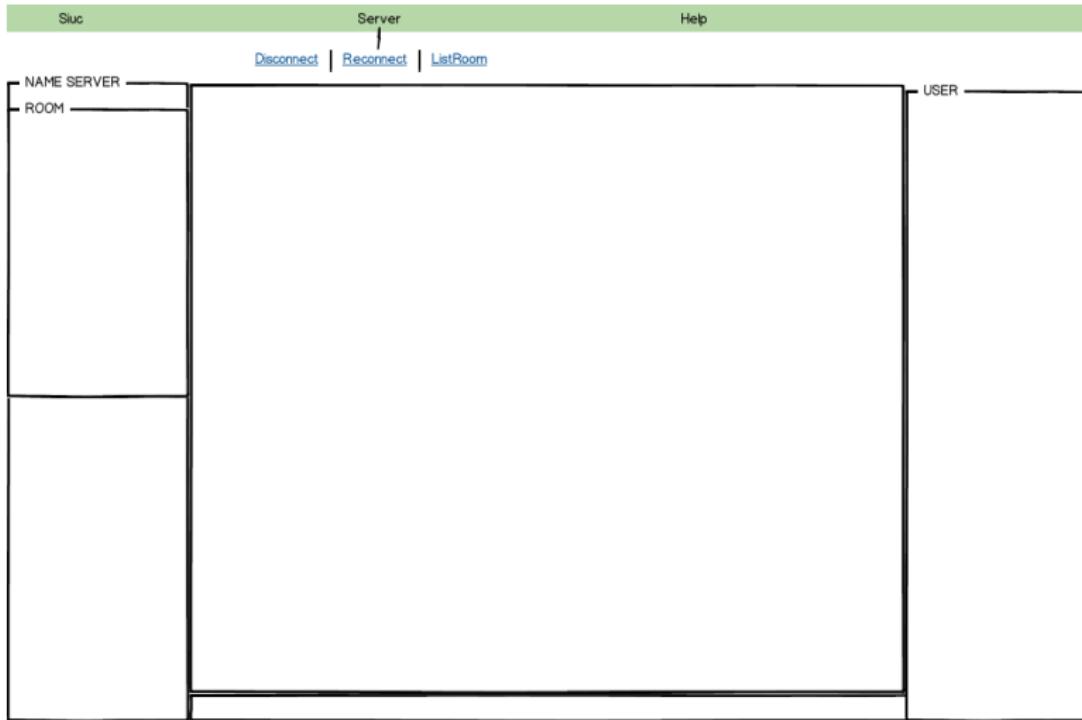


Iterazione 1: Requisiti - Prototipo User Interface



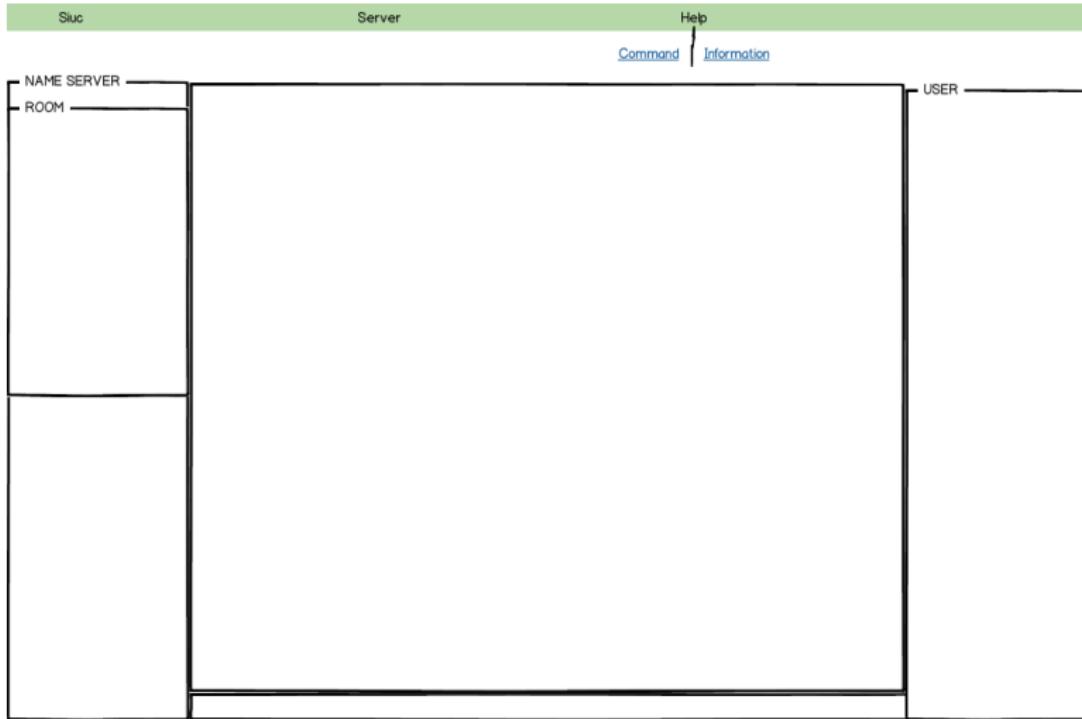


Iterazione 1: Requisiti - Prototipo User Interface



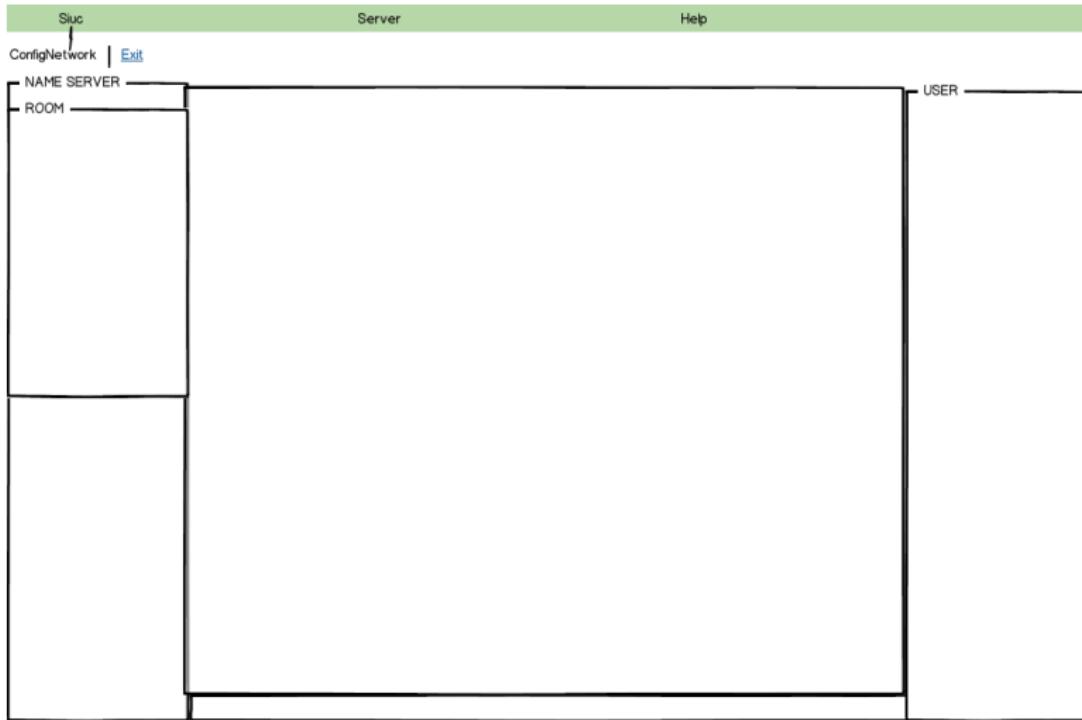


Iterazione 1: Requisiti - Prototipo User Interface





Iterazione 1: Requisiti - Prototipo User Interface





Iterazione 1: Requisiti - Prototipo User Interface

The diagram illustrates a user interface prototype for Iteration 1. It features a main window titled "Network" with the following fields:

- User:**
 - Nick name: [Text input field]
 - Real name: [Text input field]
- Server:**
 - Name: [Text input field]
 - Address: [Text input field]
 - Port: [Text input field]

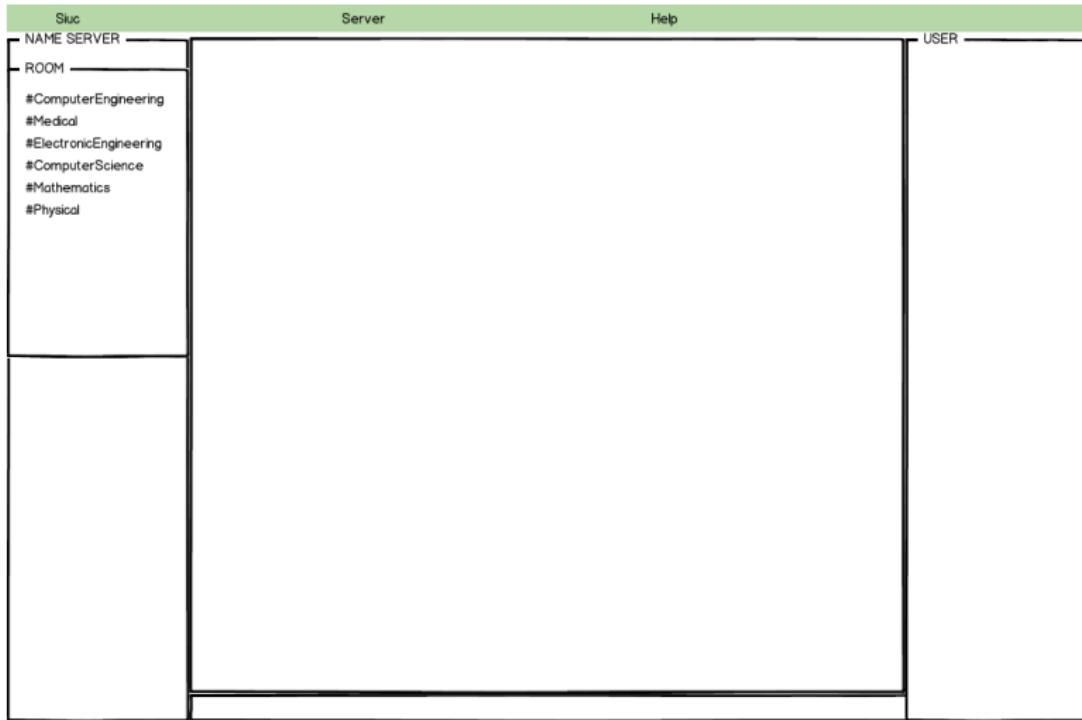
At the bottom of the window are two buttons: "Close" and "Connect".

The interface is divided into sections:

- Left Column:** Contains sections for "NAME SERVER" and "ROOM", each with a dropdown menu icon.
- Right Column:** Contains a section labeled "USER".
- Top Bar:** Includes a "Sluc" button, a "ConfigNetwork" button, and an "Exit" link.
- Bottom Bar:** A horizontal bar spanning the width of the interface.



Iterazione 1: Requisiti - Prototipo User Interface





Iterazione 1: Requisiti - Prototipo User Interface

Sluc	Server	Help
<ul style="list-style-type: none"> - NAME SERVER - ROOM <ul style="list-style-type: none"> #ComputerEngineering #Medical #ElectronicEngineering #ComputerScience #Mathematics #Physical 	<p>19 Gennaio 2015</p> <p>[ora:minuti:secondi] nicola : ciao [ora:minuti:secondi] leandro : funziona ? [ora:minuti:secondi] daniele : bellissima [ora:minuti:secondi] orazio : è un ottimo modello UP...</p>	<p>USER</p> <ul style="list-style-type: none"> - o nicola - leandro - daniele - orazio



Iterazione 1: Requisiti - Prototipo User Interface

Sluc 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 #Medical #ElectronicEngineering #ComputerScience #Mathematics #Physical	<p align="center">19 Gennaio 2015</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"> o user_medical 1 - user_medical 2 - user_medical 3 - user_medical 4





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

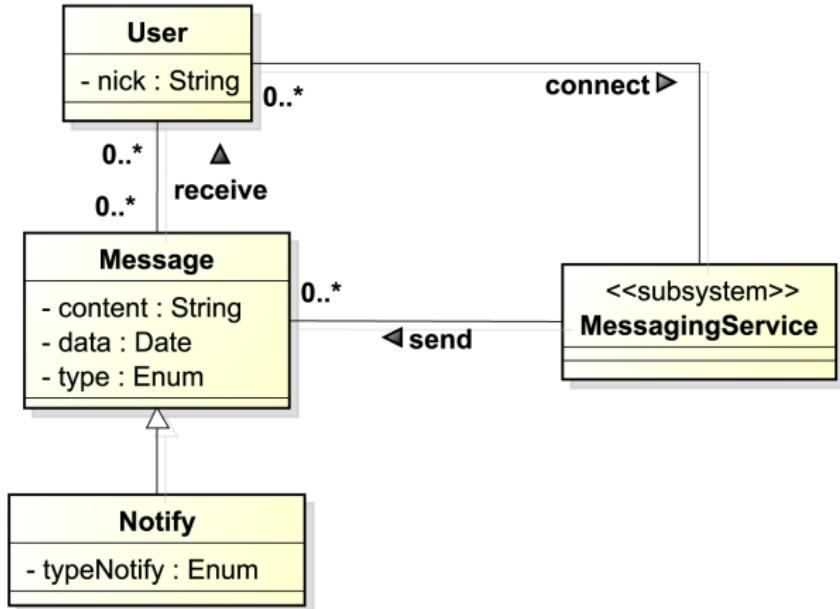


Figura 5 : UC1 - Modello di dominio

Iterazione 1: Analisi - UC1_RequestConnection

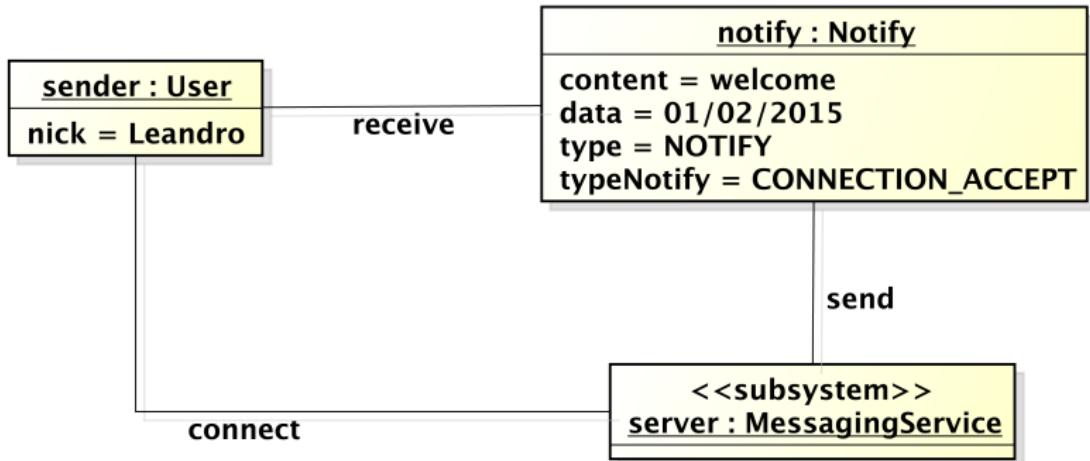


Figura 6 : UC1 - Oggetti di dominio

Iterazione 1: Analisi - UC1_RequestConnection

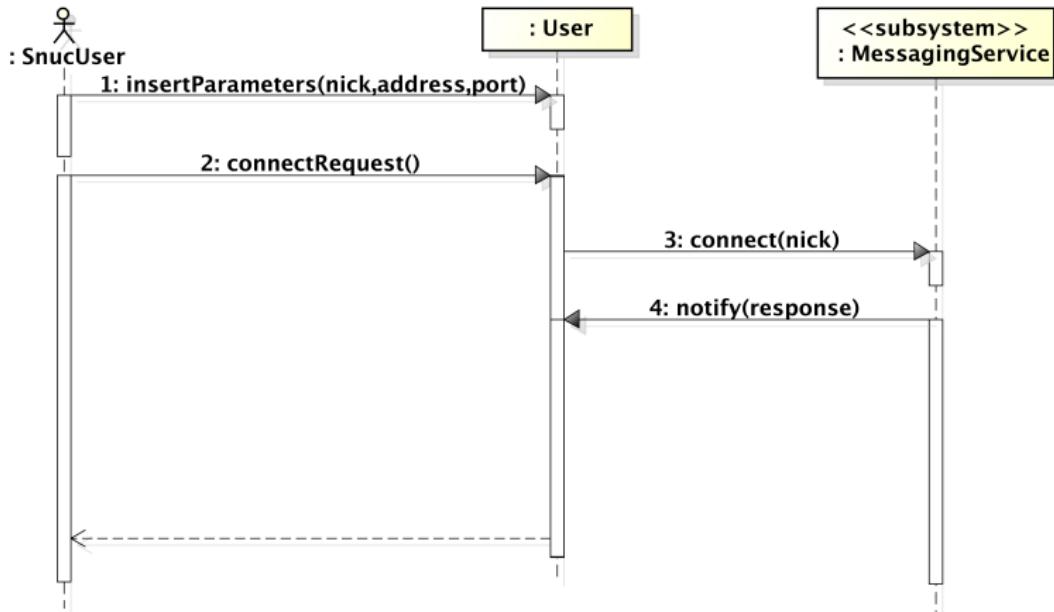


Figura 7 : UC1 - Diagramma di sequenza di dominio



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. CONNECTION_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'* è 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

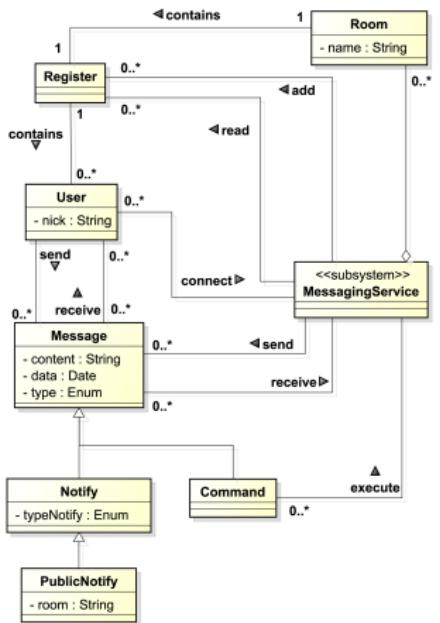


Figura 8 : UC2 - Modello di dominio

Iterazione 1: Analisi - UC2_AccessRoom

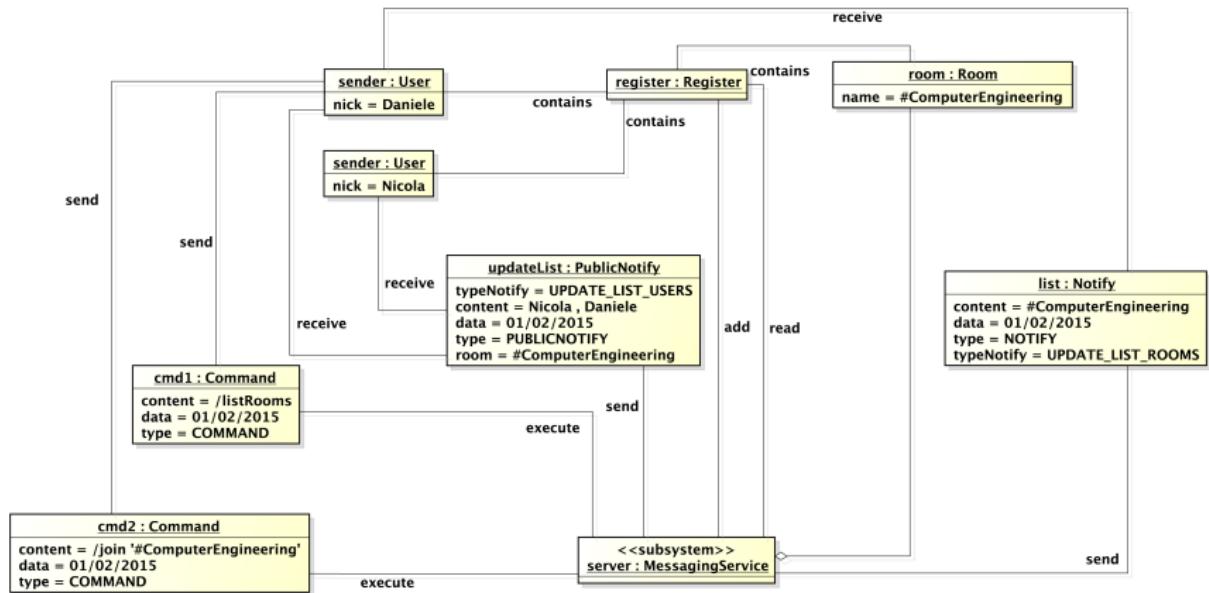


Figura 9 : UC2 - Oggetti di dominio

Iterazione 1: Analisi - UC2_AccessRoom

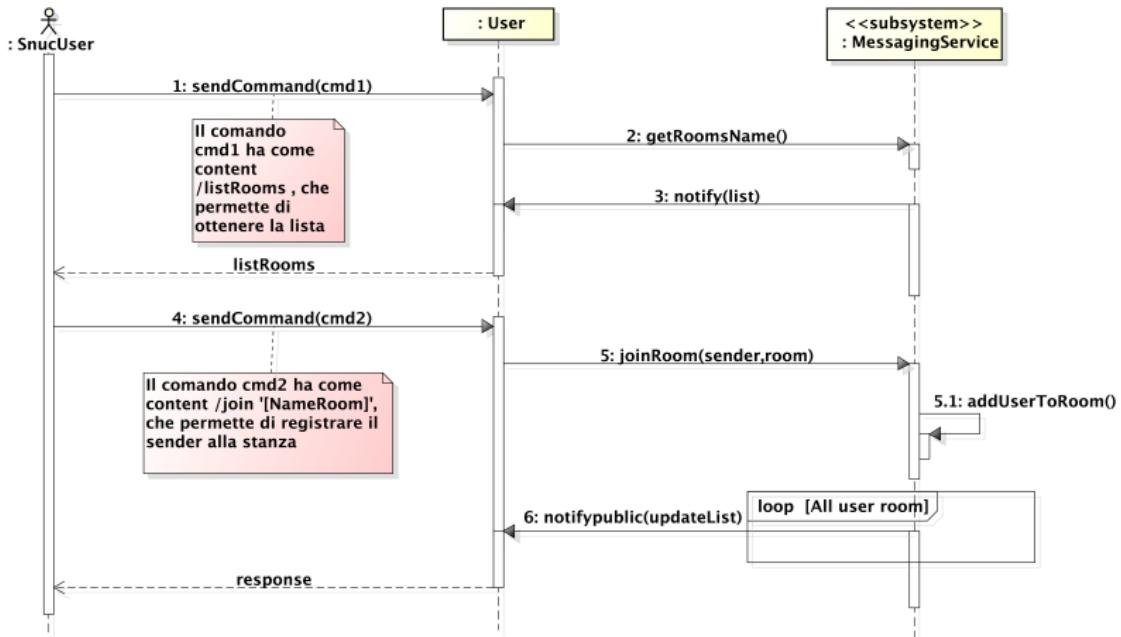


Figura 10 : UC2 - Diagramma di sequenza di dominio



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"> ▶ L'utente è connesso al servizio di messaggistica ▶ L'utente ha inserito il comando relativo alla registrazione nella stanza
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: UC2_AccessRoom
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



Iterazione 1: Progettazione - Class Diagram UC1 e UC2

Da un'analisi del modello di domino sono state realizzate le seguenti classi:

PACKAGE COMMON

- ▶ **Message:** è una classe astratta che definisce tutti gli attributi di in un generico messaggio che saranno necessari per la corretta gestione. È costituito da un “content” (contenuto del messaggio), da una “date” (rappresenta la data) e dal “sender” (mittente).
- ▶ **Command:** è una sottoclasse di Message e permette di identificare che nel contenuto del messaggio avremo un comando.



Iterazione 1: Progettazione - Class Diagram UC1 e UC2

- ▶ **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.
- ▶ **PublicNotify:** è una sottoclassificazione 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 nel quale 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 PublicNotify di comprendere a quale stanza è riferita.



Iterazione 1: Progettazione - Class Diagram UC1 e UC2

- ▶ **IUserInteraction**: è un'interfaccia che definisce tutti i metodi che saranno necessari per l'interazione tra UI e il UserController.
- ▶ **IMessagingService**: è un'interfaccia che definisce la struttura per le funzioni che dovrà svolgere il MessagingService.
- ▶ **IUser**: è un'interfaccia che definisce la struttura per le funzioni che dovrà svolgere l'UserController.





Iterazione 1: Progettazione - Class Diagram Common UC1 e UC2

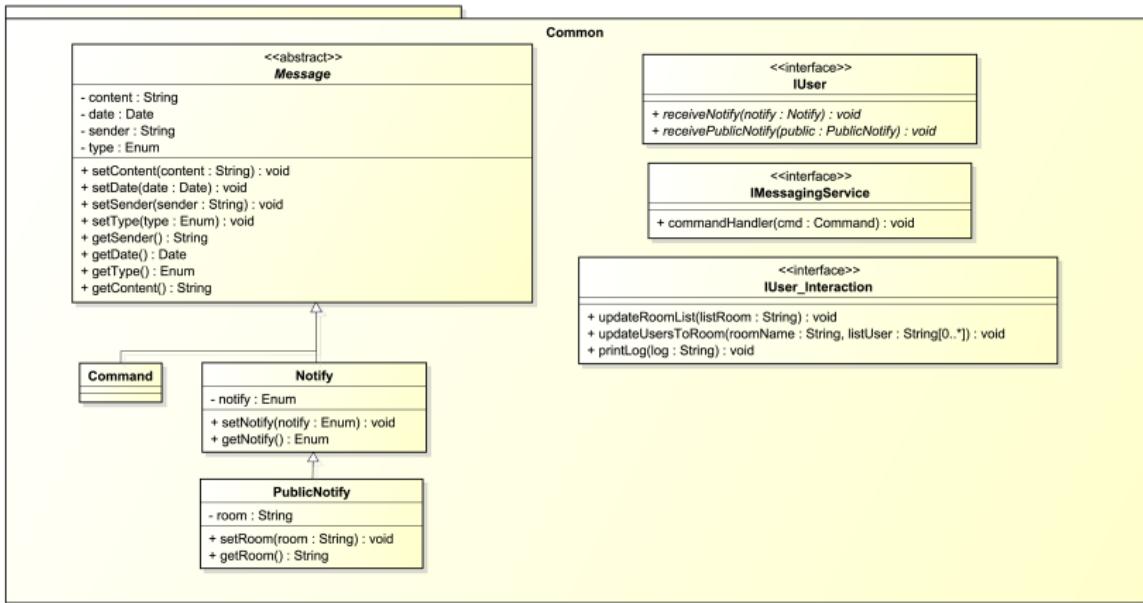


Figura 11 : DCD - Diagramma delle Classi: Package Common



Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

PACKAGE SNUC

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



Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

- ▶ **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 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.

Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

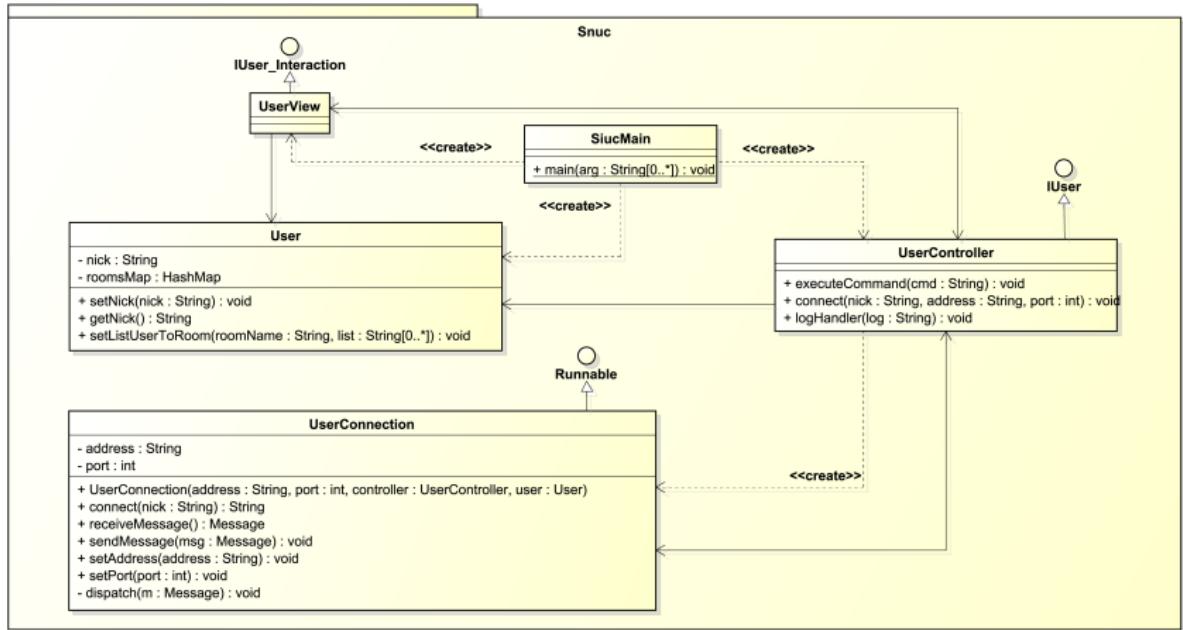


Figura 12 : DCD - Diagramma delle Classi: Package Snuc



Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

PACKAGE SNUCSERVER

- ▶ **MessagingService**: rappresenta il servizio di messaggistica ed in tale classe sono implementate tutte le funzioni necessarie al funzionamento del suddetto servizio tra cui per esempio la gestione dell'invio delle notifiche agli User, l'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.



Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

- ▶ **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.



Iterazione 1: Progettazione - Class Diagram Snuc UC1 e UC2

- ▶ **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 Snuc Server UC1 e UC2

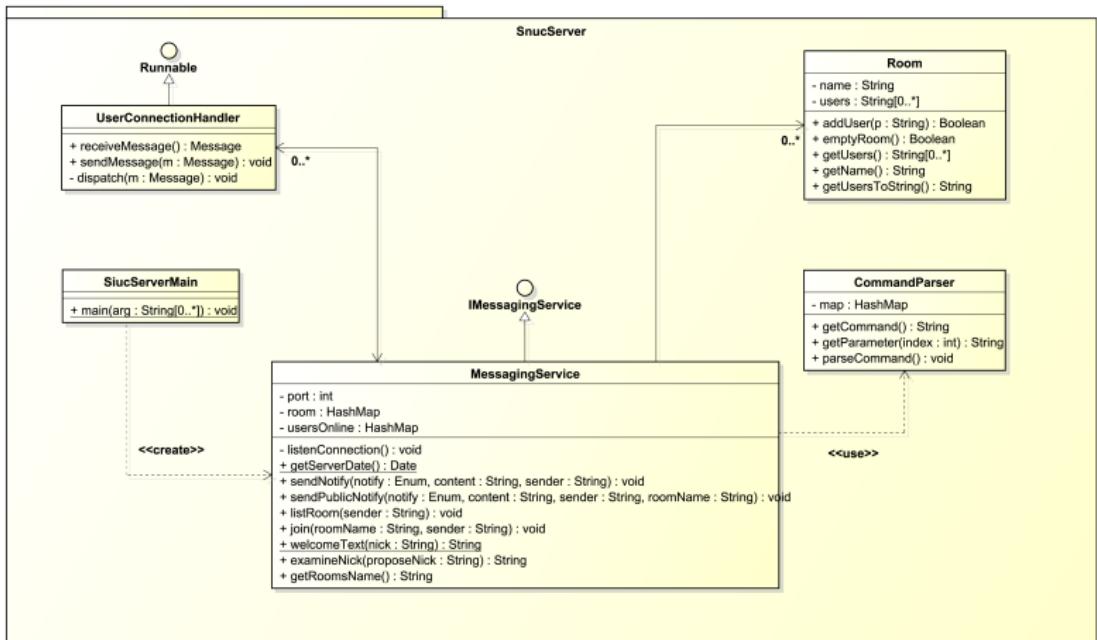


Figura 13 : DCD - Diagramma delle Classi: Package Snuc Server

Iterazione 1: Progettazione, UC1_RequestConnection

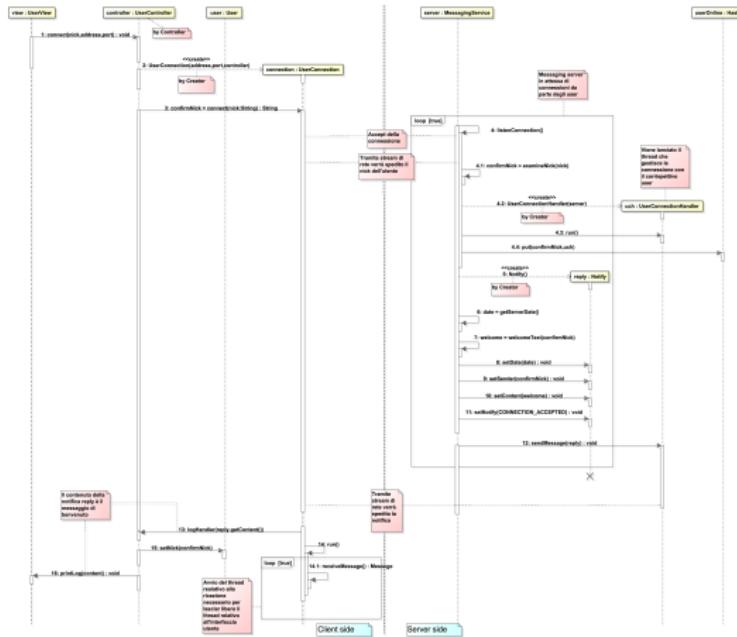


Figura 14 : SSD - OP1-4: connect(nick), notify(response) del modello dominio (figura 7)

Iterazione 1: Progettazione, UC2_AccessRoom - OP1

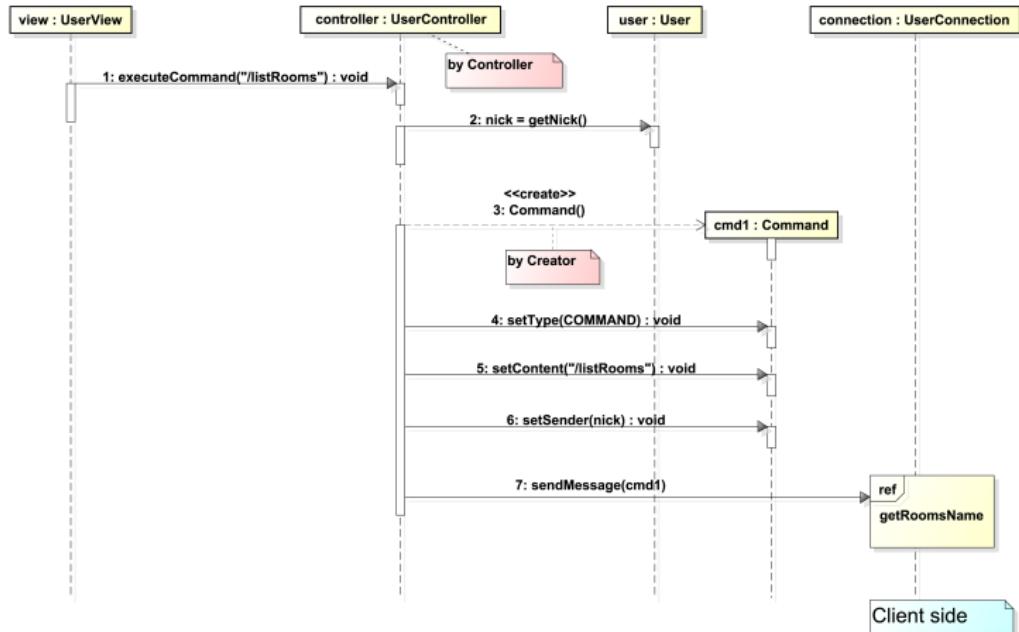


Figura 15 : SSD - OP1: sendCommand(cmd1) del modello di dominio (figura 10)

Iterazione 1: Progettazione, UC2_AccessRoom - OP2

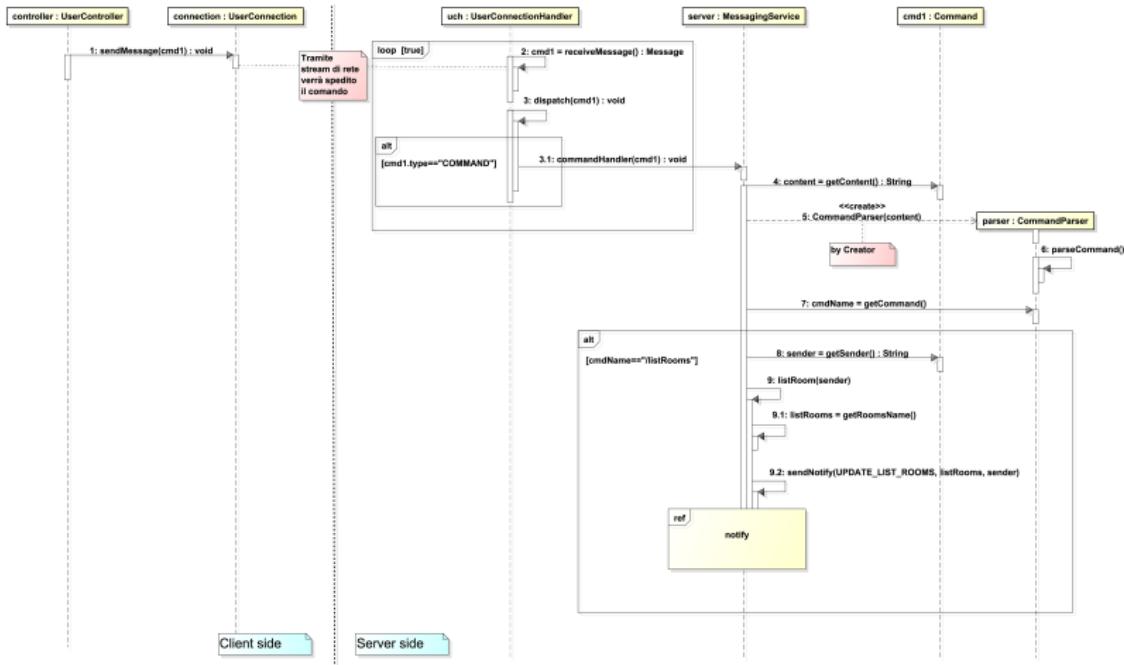


Figura 16 : SSD - OP2: `getRoomsName()` del modello di dominio (figura 10)

Iterazione 1: Progettazione, UC2_AccessRoom - OP3

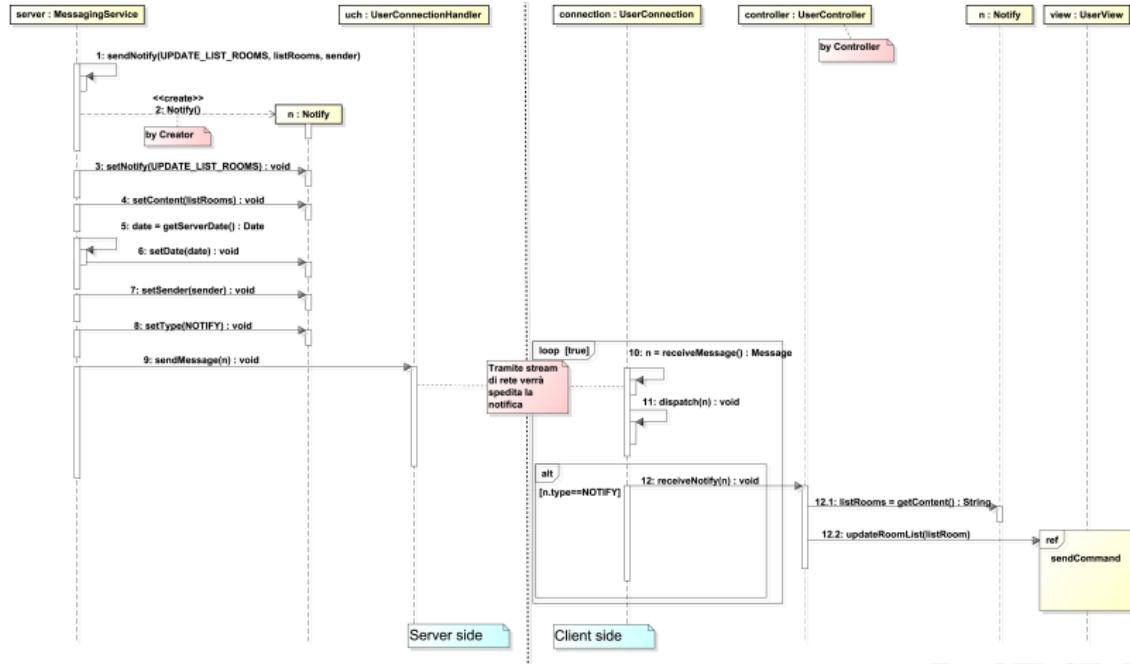


Figura 17 : SSD - OP3: notify(list) del modello di dominio (figura 10)

Iterazione 1: Progettazione, UC2_AccessRoom - OP4

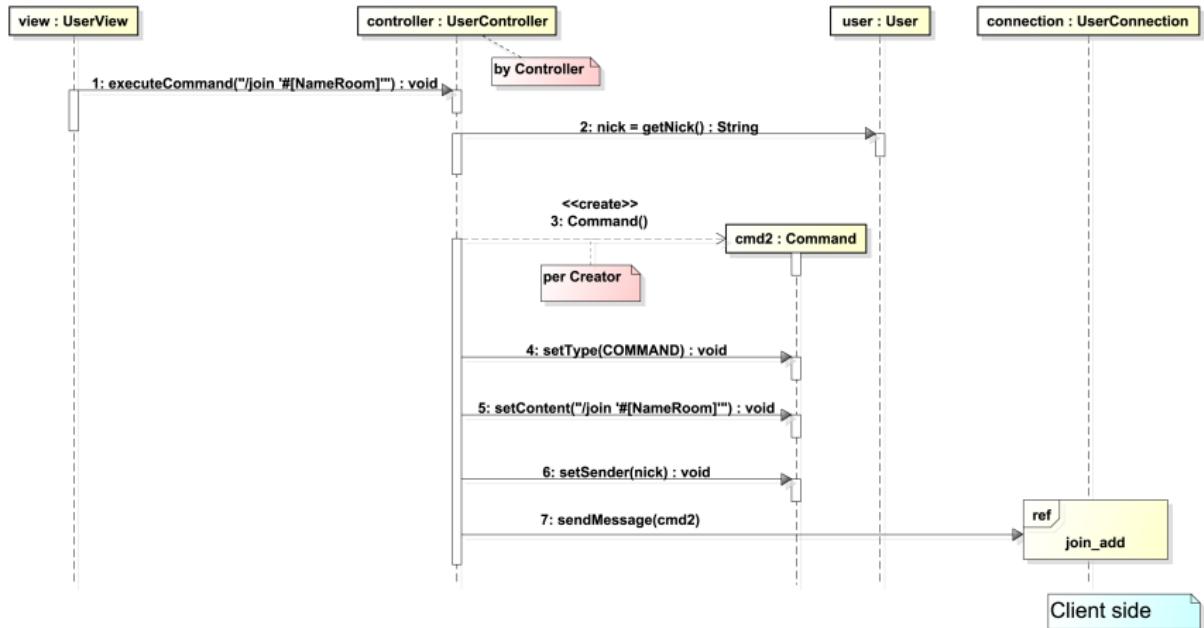


Figura 18 : SSD - OP4: sendCommand(cmd2) del modello di dominio (figura 10)

Iterazione 1: Progettazione, UC2_AccessRoom - OP5

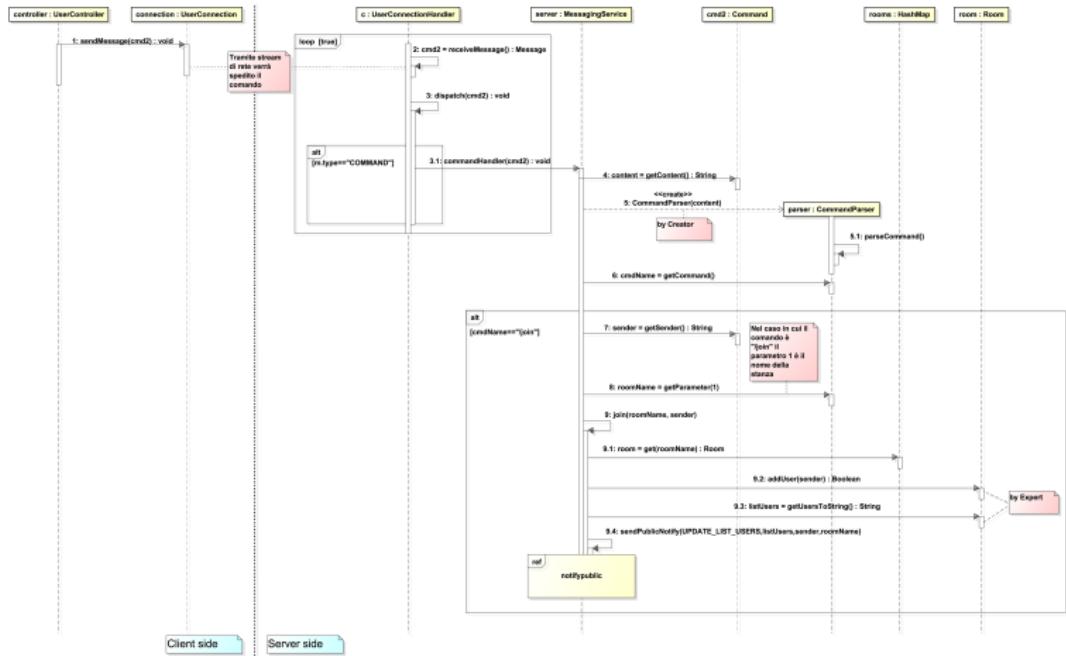


Figura 19 : SSD - OP5: joinRoom(sender,room), addUserToRoom() del modello di dominio (figura 10)

Iterazione 1: Progettazione, UC2_AccessRoom - OP6

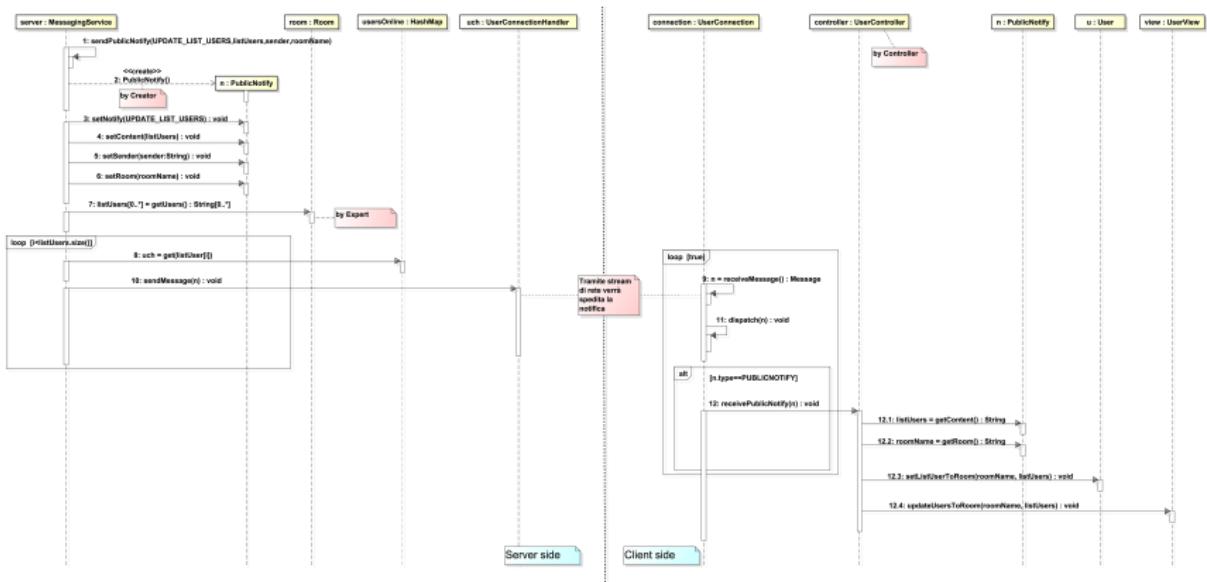


Figura 20 : SSD - OP6: `notifypublic(updateList)` del modello di dominio (figura 10)



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.



Descrizione: Refactoring Iterazione 1

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

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



Descrizione: Refactoring Iterazione 1

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 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



Descrizione: Refactoring Iterazione 1

detto rappresenterà il nostro User in locale.

In base al tipo di protocollo utilizzato ogni oggetto di cui si è parlato precedentemente 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 utilizzassimo la comunicazione UDP:

- ProxyMessagingServiceUDP
- ProxyUserUDP
- DispatcherUDP





Descrizione: Refactoring Iterazione 1

- ConnectionHandlerUDP

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



Descrizione: Refactoring Iterazione 1

e ConnectionFactoryUDP) relative al protocollo utilizzato. Tali factory implementano i 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



Descrizione: Refactoring Iterazione 1

gli oggetti che saranno osservati sono la Room e l>UserOnLine (classe che rappresenta 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 all'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

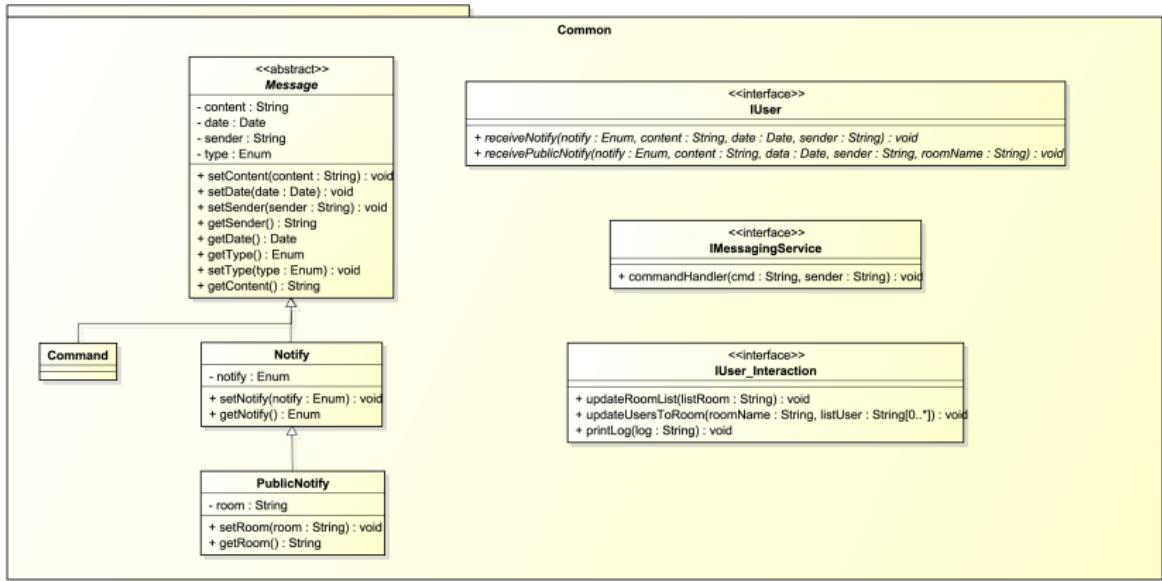


Figura 21 : DCD - Diagramma delle Classi: Package Common

Refactoring Iterazione 1: Class Diagram Snuc UC1 e UC2

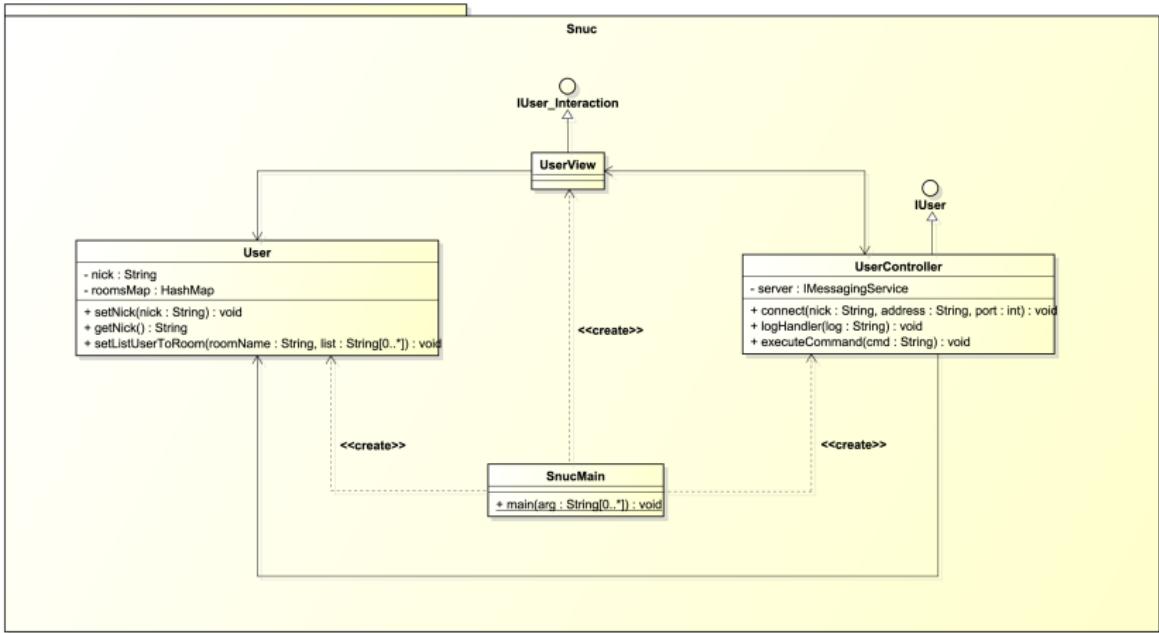


Figura 22 : DCD - Diagramma delle Classi: Package Snuc



Refactoring Iterazione 1: Class Diagram Connector UC1 e UC2

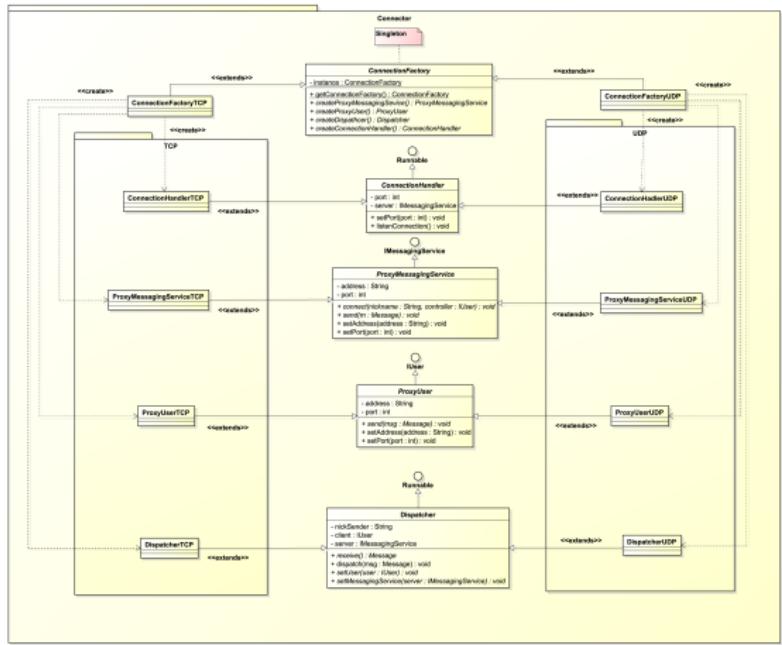


Figura 23 : DCD - Diagramma delle Classi: Package Connector

Refactoring Iterazione 1: Class Diagram Snuc Server UC1 e UC2

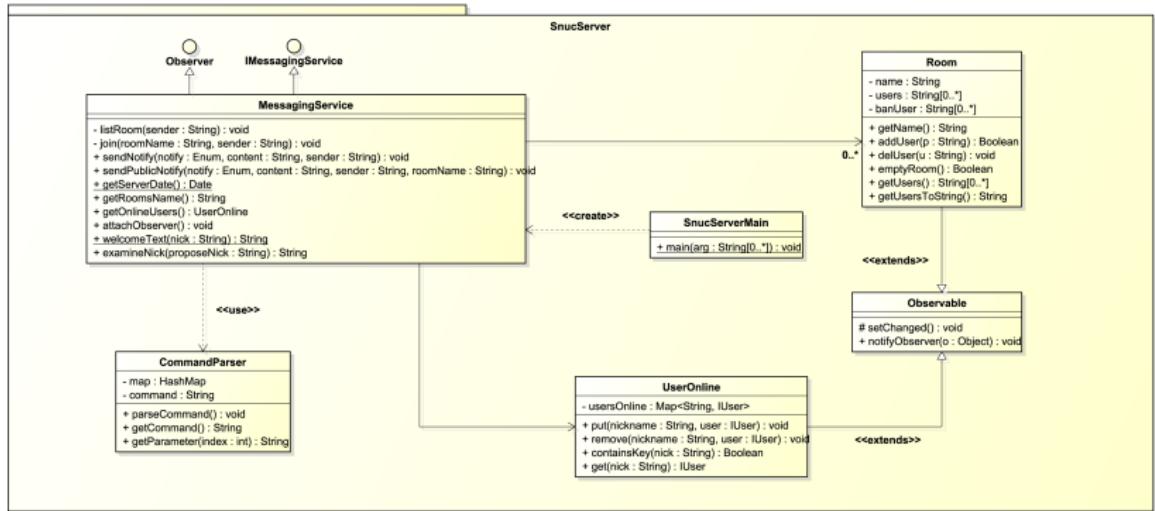


Figura 24 : DCD - Diagramma delle Classi: Package Snuc Server



Refactoring Iterazione 1: UC1_RequestConnection

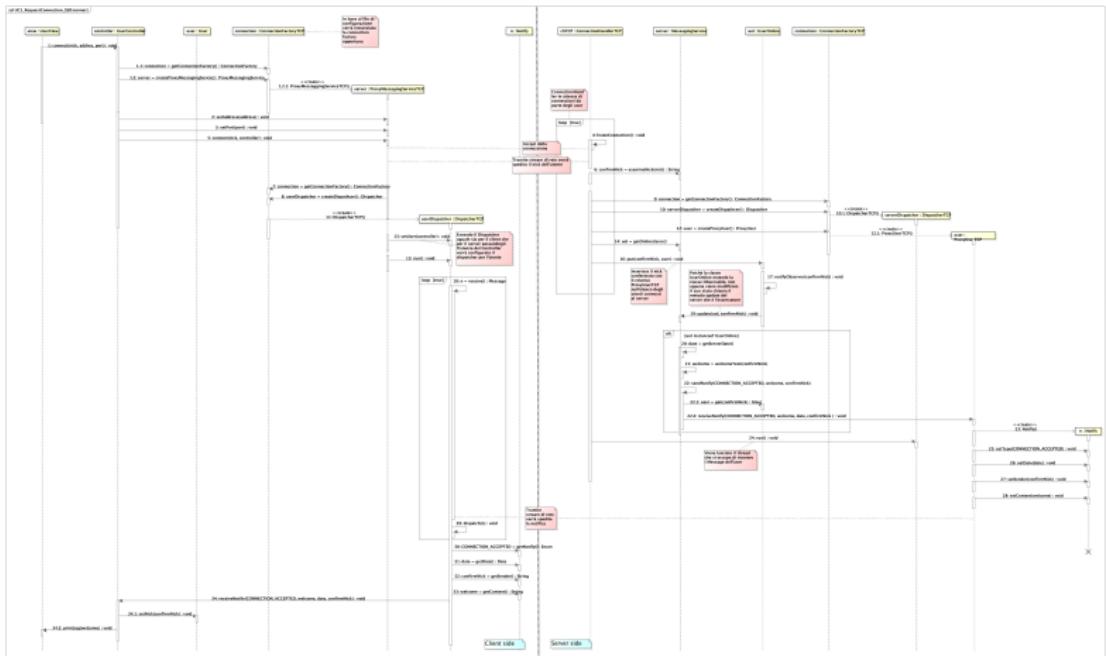


Figura 25 : SSD - OP1-4: connect(nick), notify(response) del modello dominio (figura 7)

Refactoring Iterazione 1: UC2_AccessRoom - OP1

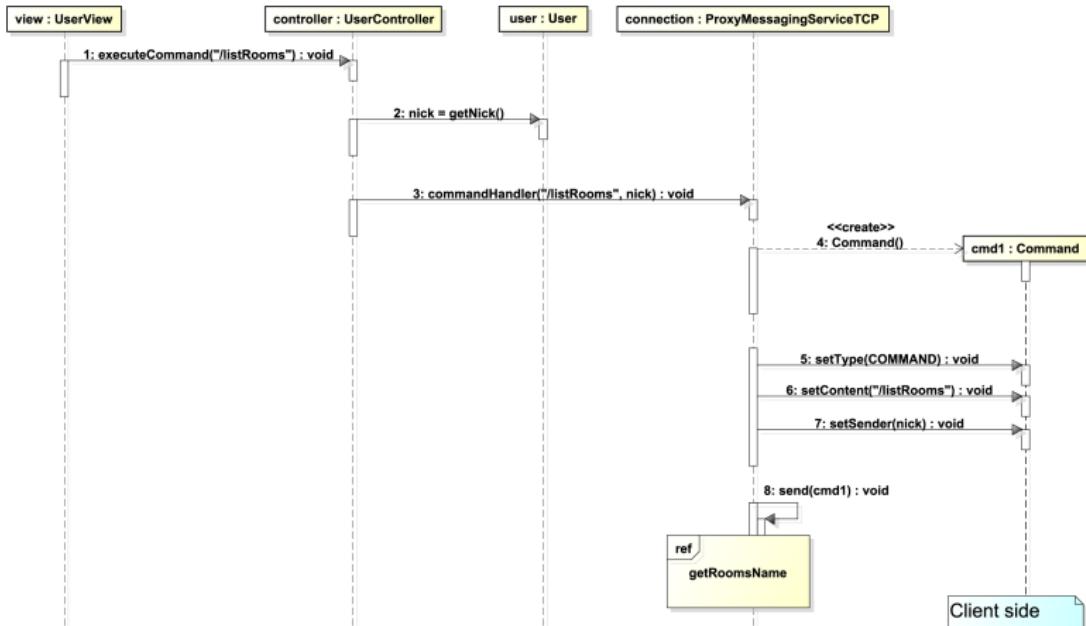


Figura 26 : SSD - OP1: sendCommand(cmd1) del modello di dominio (figura 10)



Refactoring Iterazione 1: UC2_AccessRoom - OP2

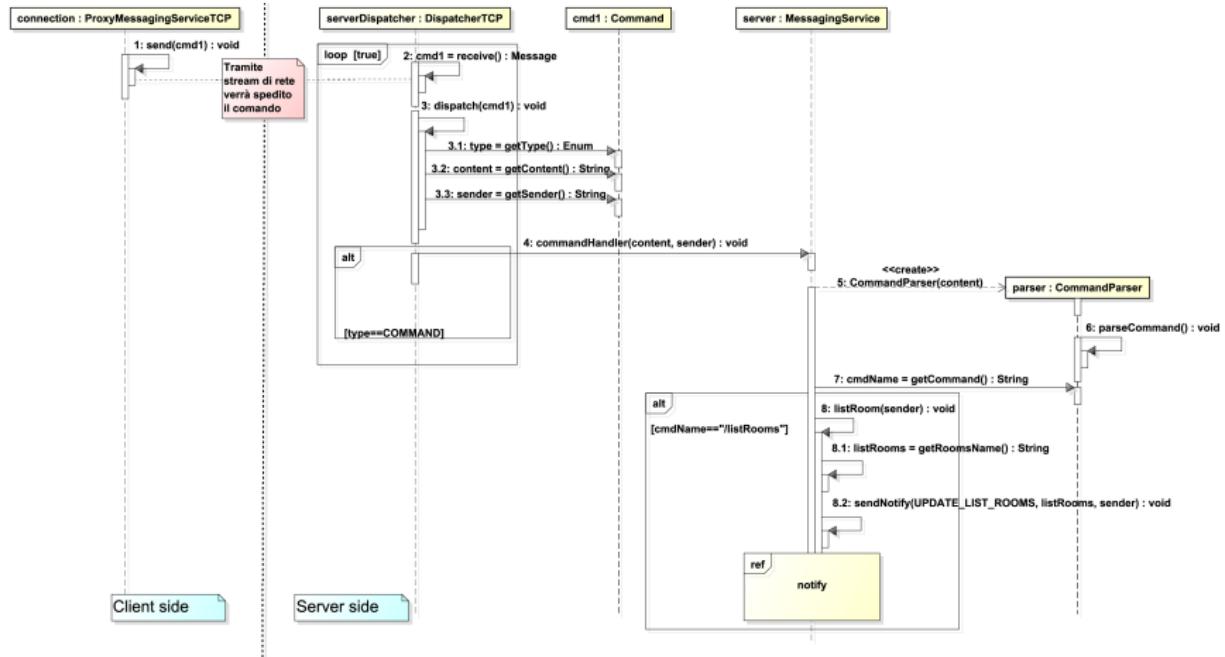


Figura 27 : SSD - OP2: getRoomsName() del modello di dominio (figura 10)

Refactoring Iterazione 1: UC2_AccessRoom - OP3

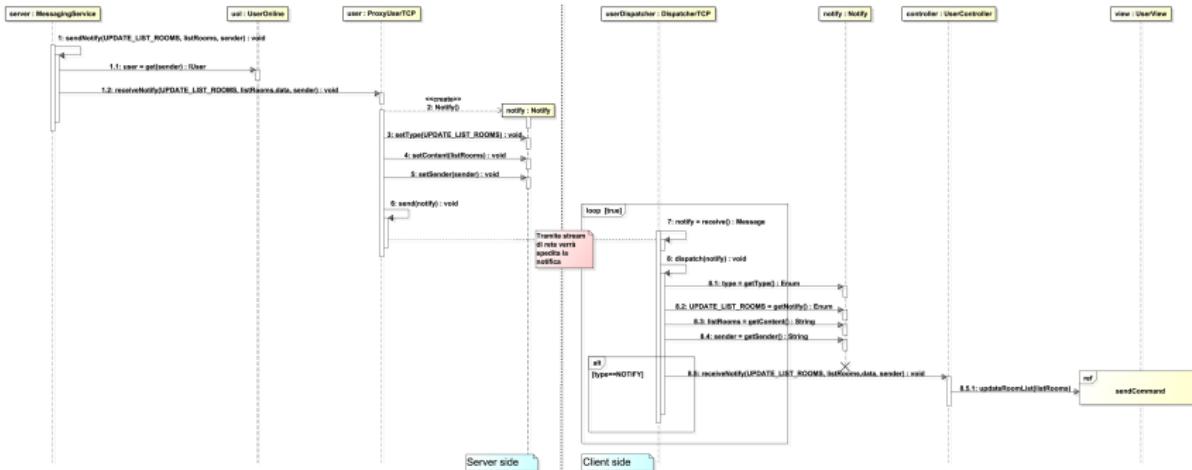


Figura 28 : SSD - OP3: notify(list) del modello di dominio (figura 10)

Refactoring Iterazione 1: UC2_AccessRoom - OP4

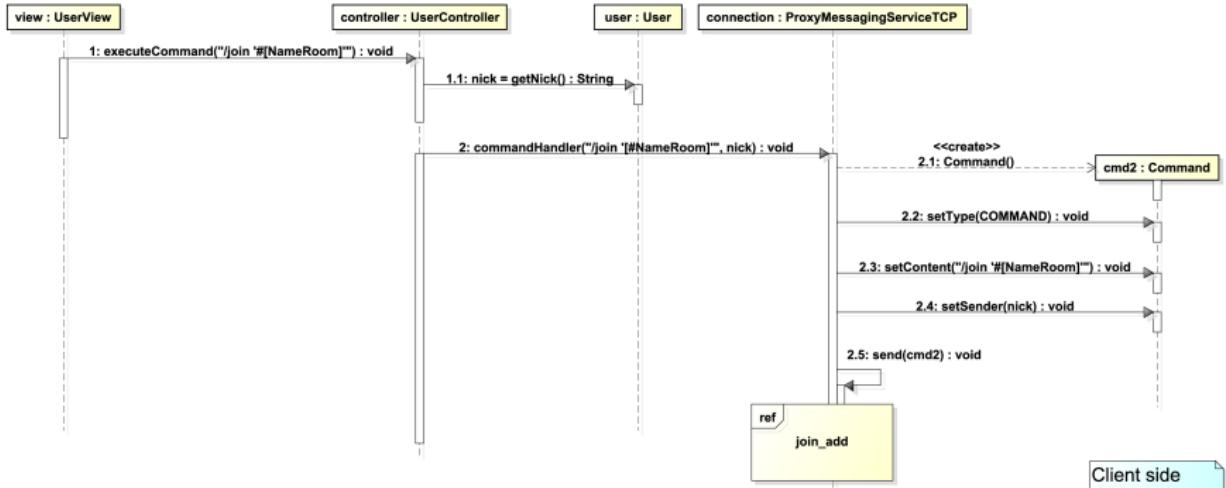


Figura 29 : SSD - OP4: sendCommand(cmd2) del modello di dominio (figura 10)



Refactoring Iterazione 1: UC2_AccessRoom - OP5

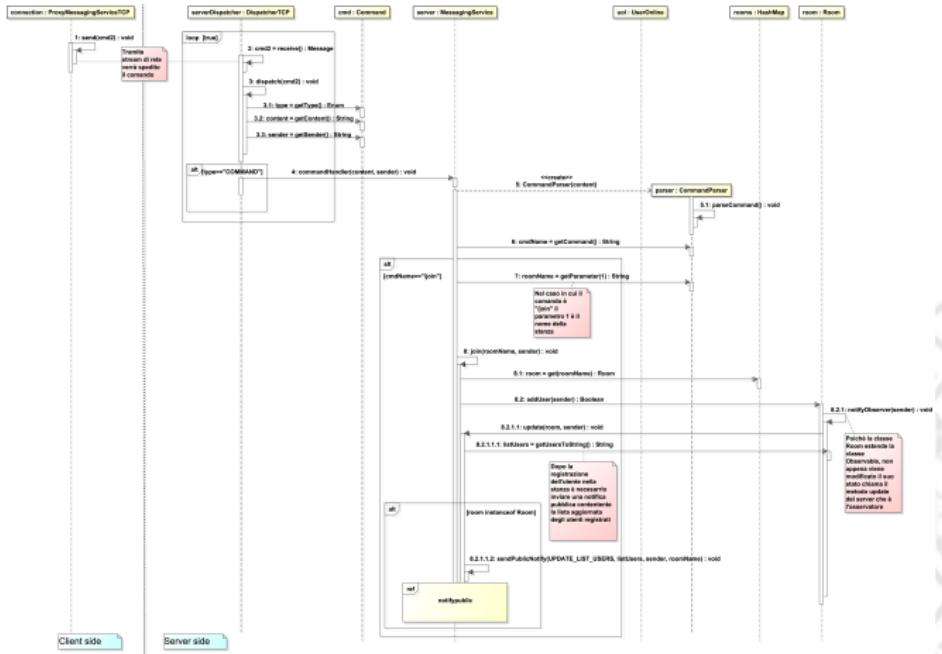


Figura 30 : SSD - OP5: joinRoom(sender,room), addUserToRoom() del modello di dominio (figura 10)

Refactoring Iterazione 1: UC2_AccessRoom - OP6

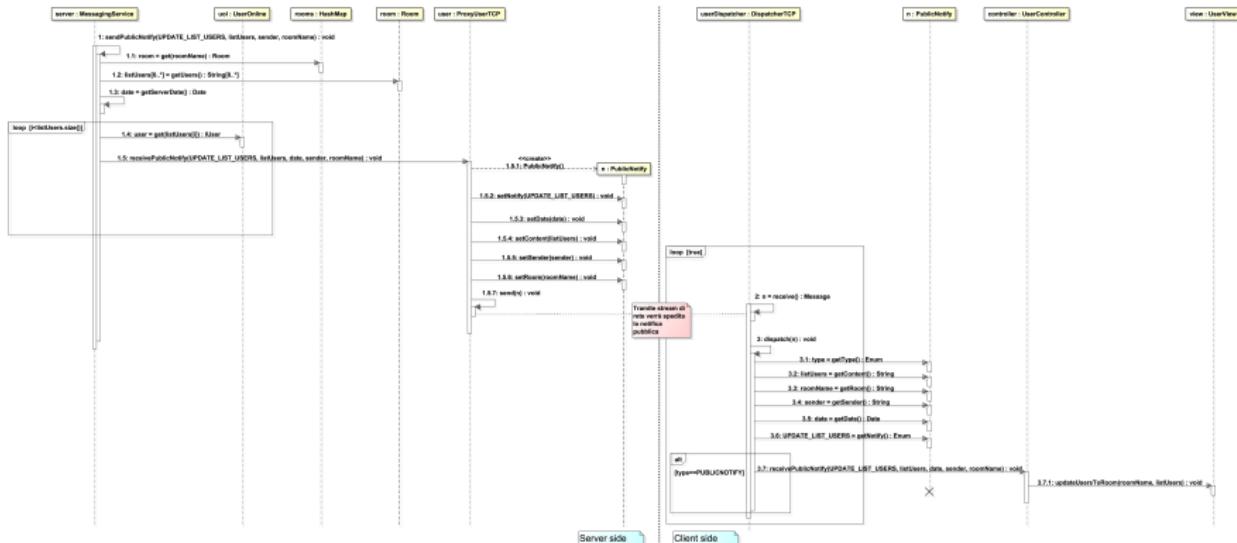


Figura 31 : SSD - OP6: `notifypublic(updateList)` del modello di dominio (figura 10)



Refactoring Iterazione 1: Descrizione Implementazione

L'implementazione del progetto è stata realizzata sfruttando il linguaggio ad oggetti Java 1.8, utilizzando l'ambiente di sviluppo NetBeans IDE 8. L'interfaccia grafica è stata realizzata tramite il framework Swing di Java. Inoltre gli screenshot di seguito rappresentano il risultato dell'implementazione effettuata per la prima iterazione e per il refactored relativa ai casi d'uso UC1_RequestConnection e UC2_AccessRoom.

Il codice relativo delle iterazione_1 e iterazione_1_refactored è reperibile nel repository GitHub del progetto.

Refactoring Iterazione 1: Implementazione UC1_RequestConnection

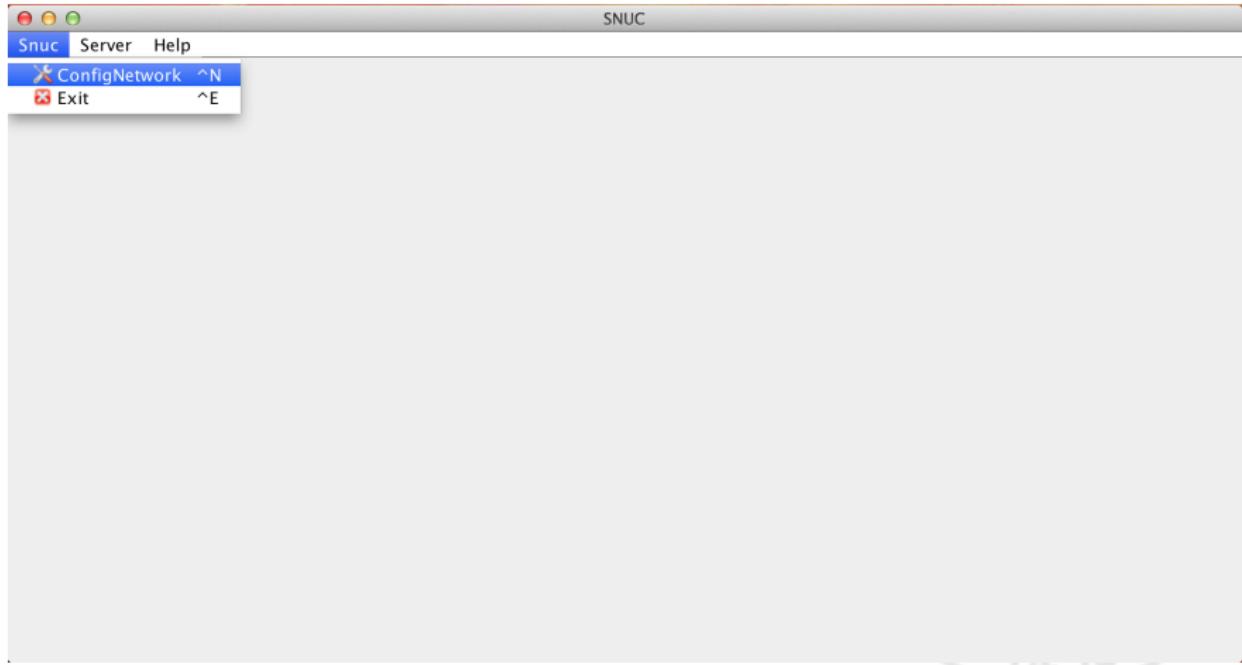


Figura 32 : UC1 - RequestConnection



Refactoring Iterazione 1: Implementazione UC1_RequestConnection

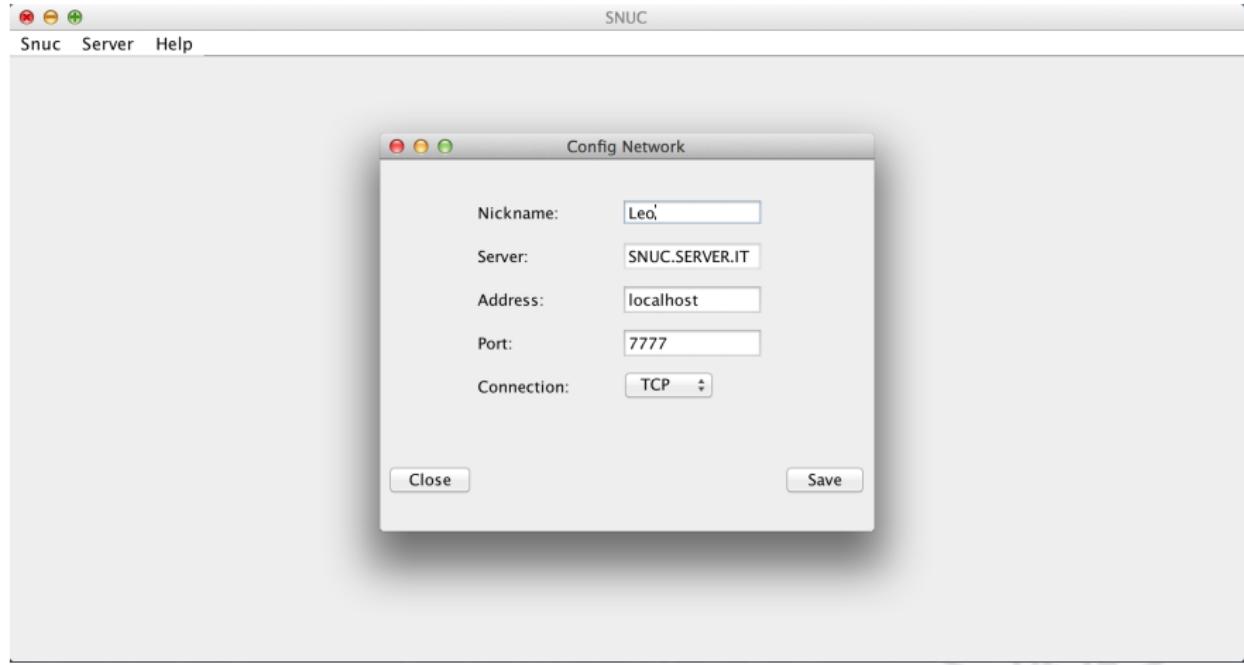


Figura 33 : UC1 - RequestConnection



Refactoring Iterazione 1: Implementazione UC1_RequestConnection

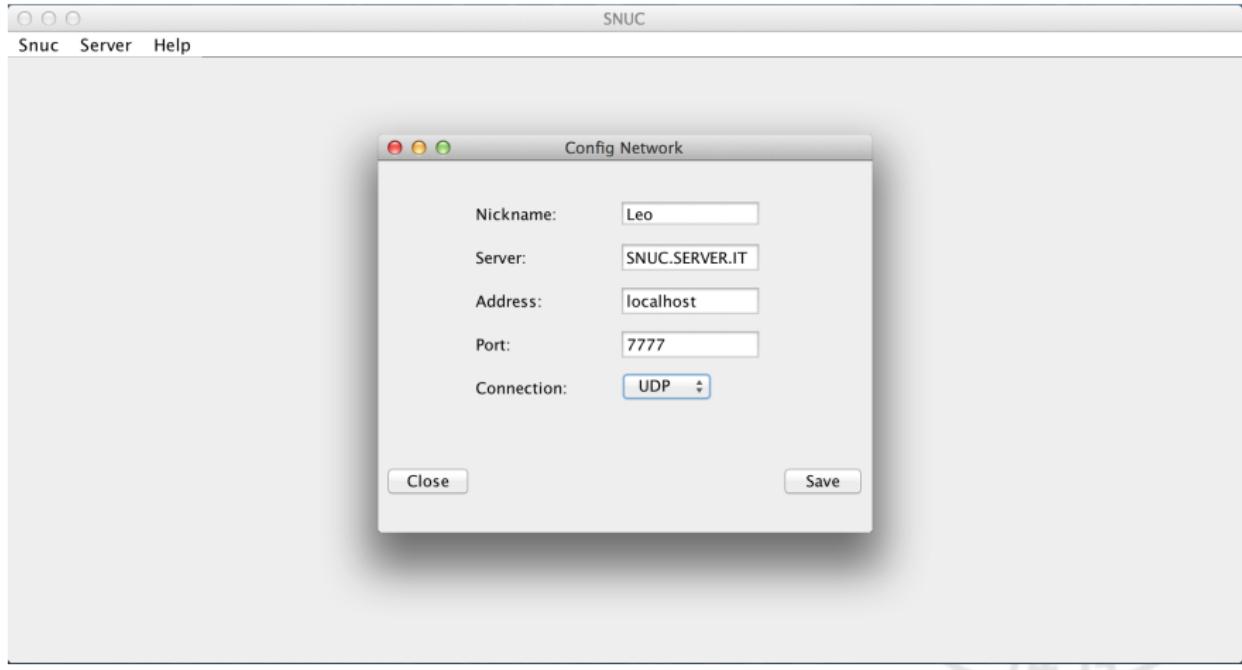


Figura 34 : UC1 - RequestConnection



Refactoring Iterazione 1: Implementazione UC1_RequestConnection

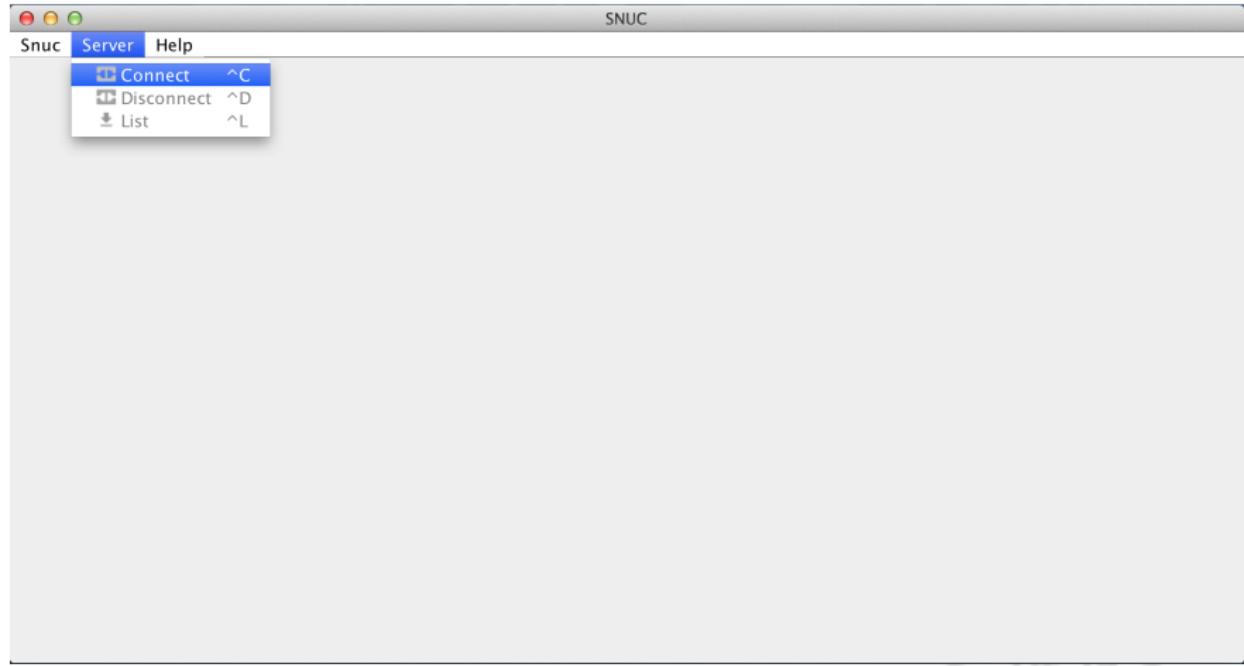


Figura 35 : UC1 - RequestConnection



Refactoring Iterazione 1: Implementazione UC1_RequestConnection

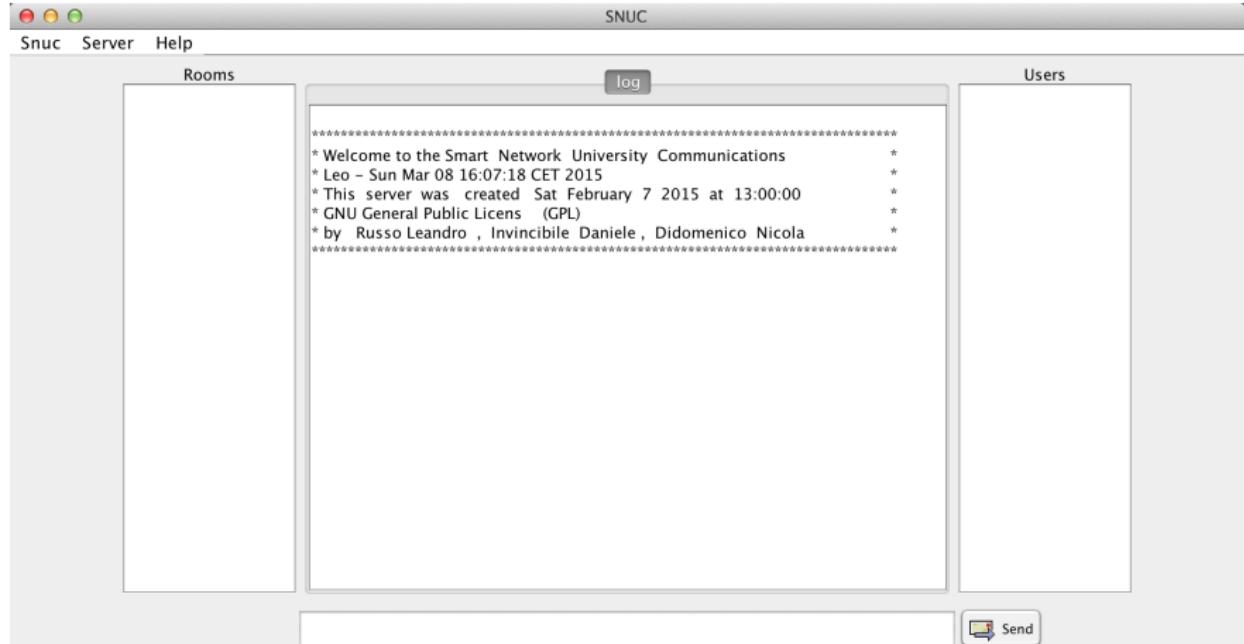


Figura 36 : UC1 - RequestConnection



Refactoring Iterazione 1: Implementazione UC2_AccessRoom

The screenshot shows a desktop application window titled "SNUC". The menu bar includes "SNUC", "Server", and "Help". The main interface is divided into three panels:

- Rooms:** A list containing "#Mathematics", "#Pharmacy", "#Medical", "#ElettronicEngineering", and "#ComputerScience".
- log:** A window displaying server logs:


```
*****
* Welcome to the Smart Network University Communications      *
* Leo - Sun Mar 08 16:07:18 CET 2015                         *
* This server was created Sat February 7 2015 at 13:00:00       *
* GNU General Public Licens (GPL)                            *
* by Russo Leandro , Invincibile Daniele , Didomenico Nicola*
*****

```
- Users:** An empty panel.

At the bottom, there is a text input field containing "/join '#Medical'" and a "Send" button.

Figura 37 : UC2 - AccessRoom

Refactoring Iterazione 1: Implementazione UC2_AccessRoom

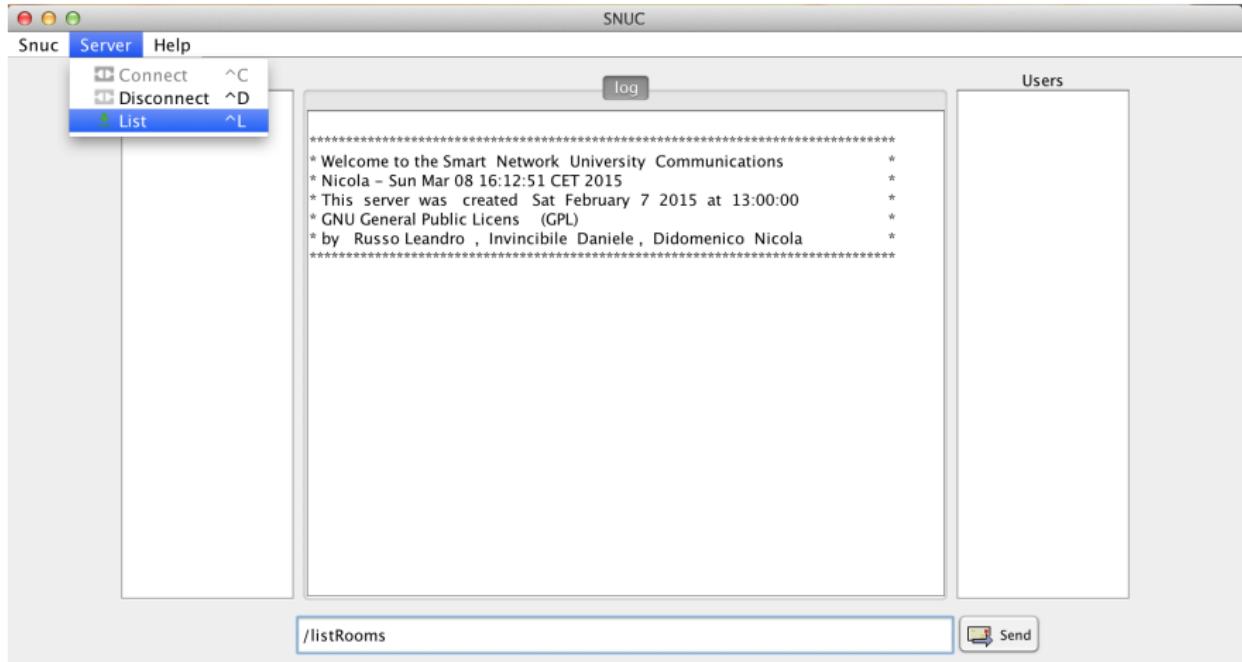


Figura 38 : UC2 - AccessRoom



Refactoring Iterazione 1: Implementazione UC2_AccessRoom

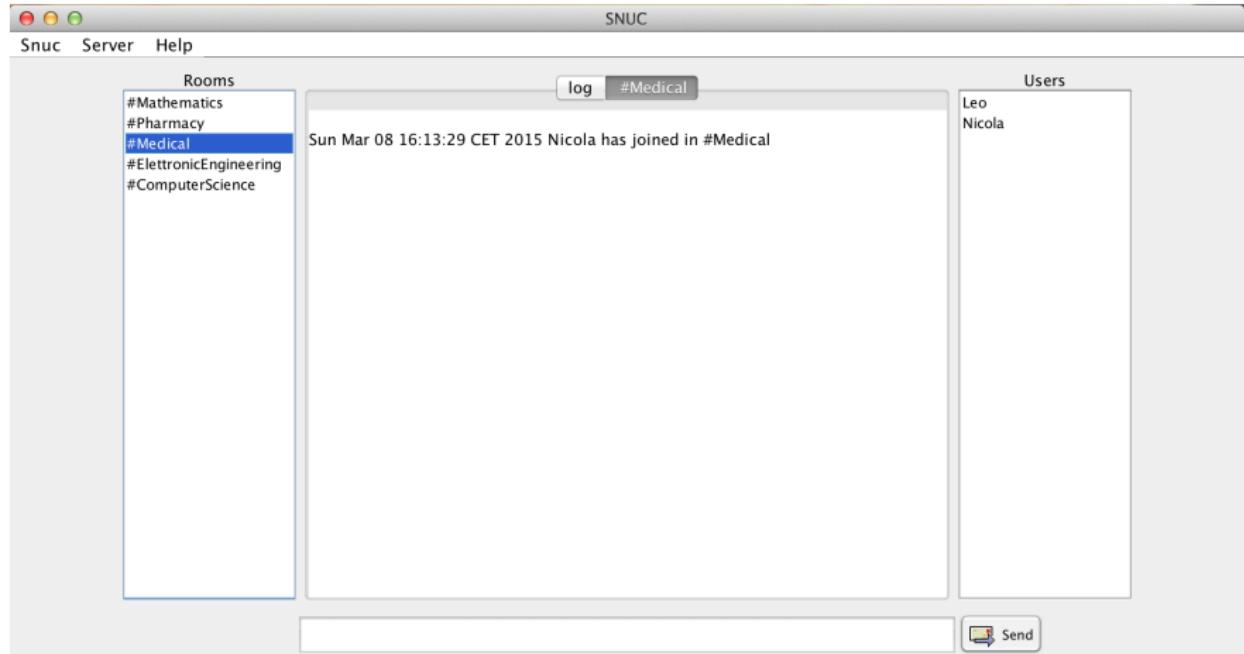


Figura 39 : UC2 - AccessRoom



Refactoring Iterazione 1: Test UC1 e UC2

Questa è l'ultima dei cinque workflows che descrive lo sviluppo del software ed è anch'essa estremamente importante e delicata. Un programma perché possa essere consegnato al committente o immesso nel mercato ha bisogno di essere garantito come funzionante. Testare tutte le possibilità di stati in cui può transitare un programma sarebbe una cosa dispendiosa e difficile per piccoli progetti, pressoché impossibile per programmi di grandi dimensioni. Ecco dunque che diventano indispensabili altri tool specifici per il testing, che possano generare casi di test e dare una mano automatizzando controlli altrimenti eccessivamente lunghi e a loro volta soggetti ad errori.

JUnit è un unit test framework per il linguaggio di programmazione



Refactoring Iterazione 1: Test UC1 e UC2

Java, introdotto per agevolare il lavoro di manutenzione e configurazione del codice.

Con unit testing si intende la procedura usata per verificare le singole parti di un codice sorgente. Se come nel nostro caso si decide di procedere al test del codice utilizzando JUnit, si devono creare delle apposite classi di test che controlleranno pezzo per pezzo il corretto funzionamento dei metodi.

Nella slide seguente possiamo visualizzare un esempio di test effettuato, sono stati testati due metodi del CommandParser, ovvero il `getCommand()` e il `getParameter ()`. Nel primo caso è stato inserito un comando appositamente errato per verificare che non ci siano malfunzionamenti, invece nel secondo test si verifica la correttezza



Refactoring Iterazione 1: Test UC1 e UC2

del parametro in relazione al comando inserito.

Ulteriori test sono reperibili nel repository GitHub del progetto.





Refactoring Iterazione 1: Test UC1 e UC2

Listing 1 : Esempio di test effettuato

```
1 public class CommandParserTest {  
2  
3     public void testGetCommand1() {  
4         System.out.println("getCommand1");  
5         CommandParser instance=new CommandParser("join '#Medical'");  
6         String result = instance.getCommand();  
7         assertNull(result);  
8     }  
9  
10    public void testgetParameter() {  
11        System.out.println("getParameter");  
12        CommandParser instance=new CommandParser("/join '#Medical'");  
13        int index = 1;  
14        String expResult = "#Medical";  
15        String result = instance.getParameter(index);  
16        assertEquals(expResult, result);  
17    }  
18 }
```



Iterazione 2: Requisiti - UC3_SendPublicMessage

Tabella 7 : Caso d'uso UC3_SendPublicMessage

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



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"> ▶ È possibile inviare messaggi confidenziali, autenticati e integri al server del servizio di messaggistica. ▶ 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 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. CONNECTION_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

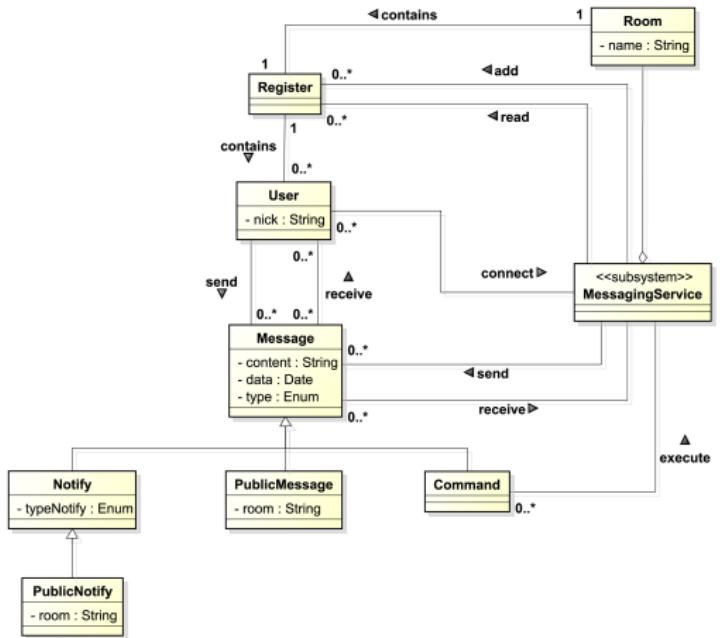


Figura 40 : UC3 - Modello di dominio

Iterazione 2: Analisi - UC3_SendPublicMessage

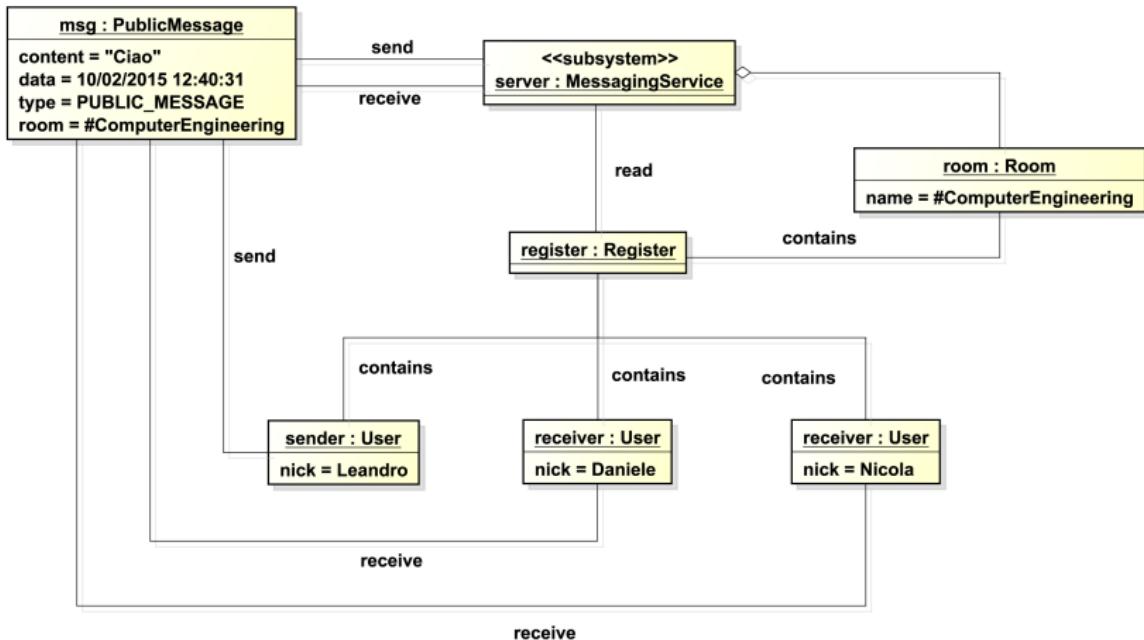


Figura 41 : UC3 - Oggetti di dominio

Iterazione 2: Analisi - UC3_SendPublicMessage

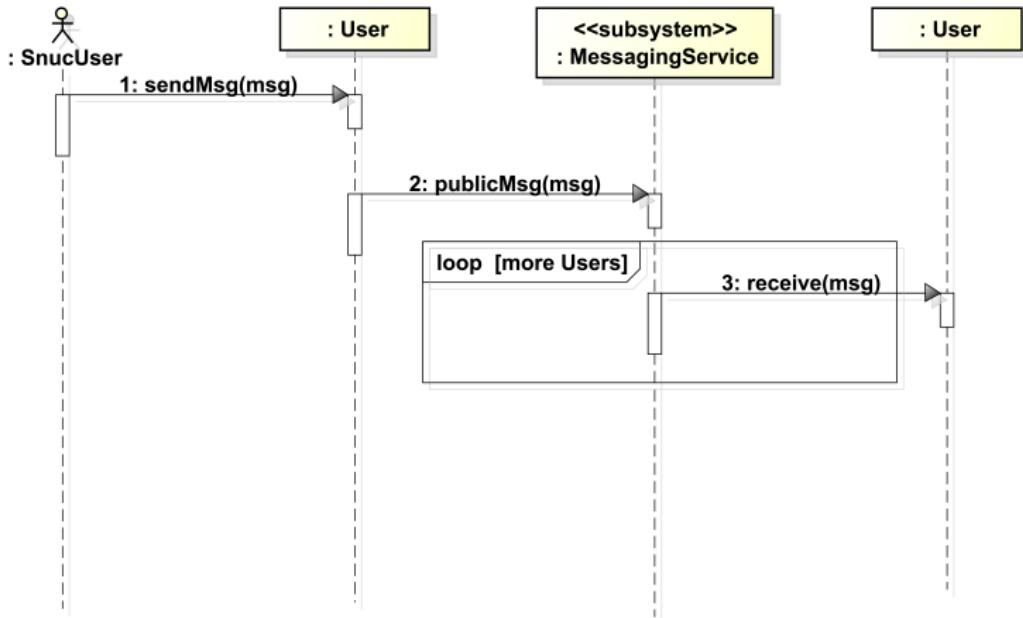


Figura 42 : UC3 - Diagramma di sequenza di dominio



Iterazione 2: Analisi, UC3 contratto CO5

Tabella 8 : UC3 Contratto CO5 - publicMsg

Operazione	<i>publicMsg(msg:PublicMessage)</i>
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





Iterazione 2: Progettazione - Descrizione

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

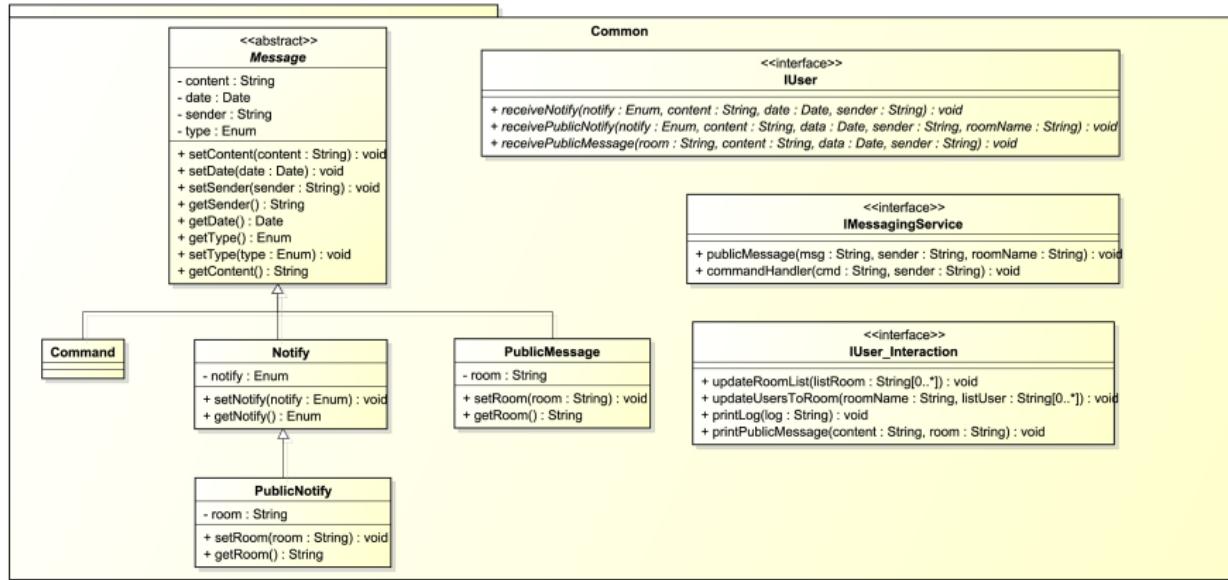


Figura 43 : DCD - Diagramma delle Classi: Package Common

Iterazione 2: Progettazione - Class Diagram Snuc UC3

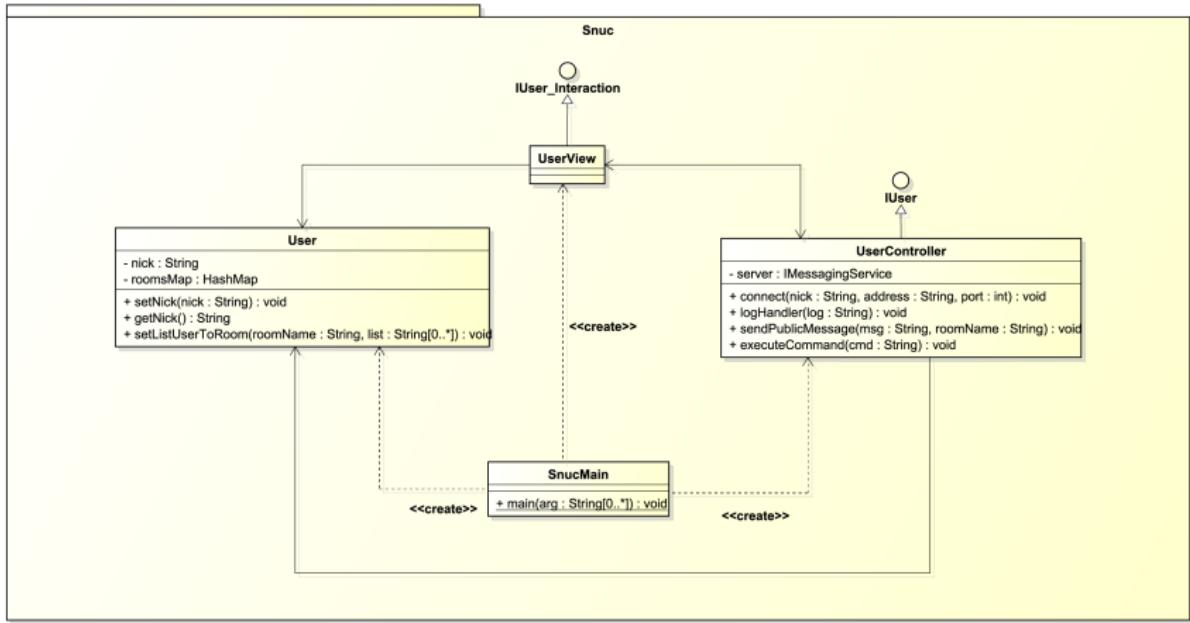


Figura 44 : DCD - Diagramma delle Classi: Package Snuc



Iterazione 2: Progettazione - Class Diagram Connector UC3

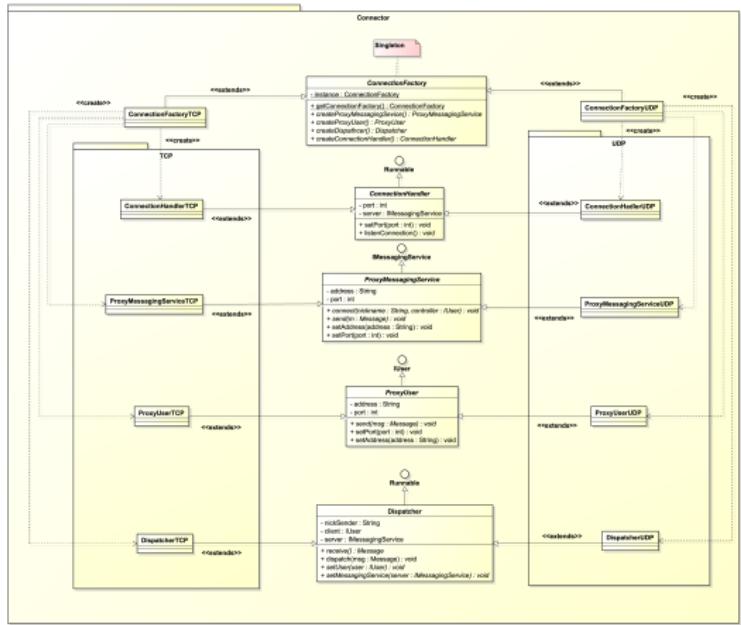


Figura 45 : DCD - Diagramma delle Classi: Package Connector

Iterazione 2: Progettazione - Class Diagram Snuc Server UC3

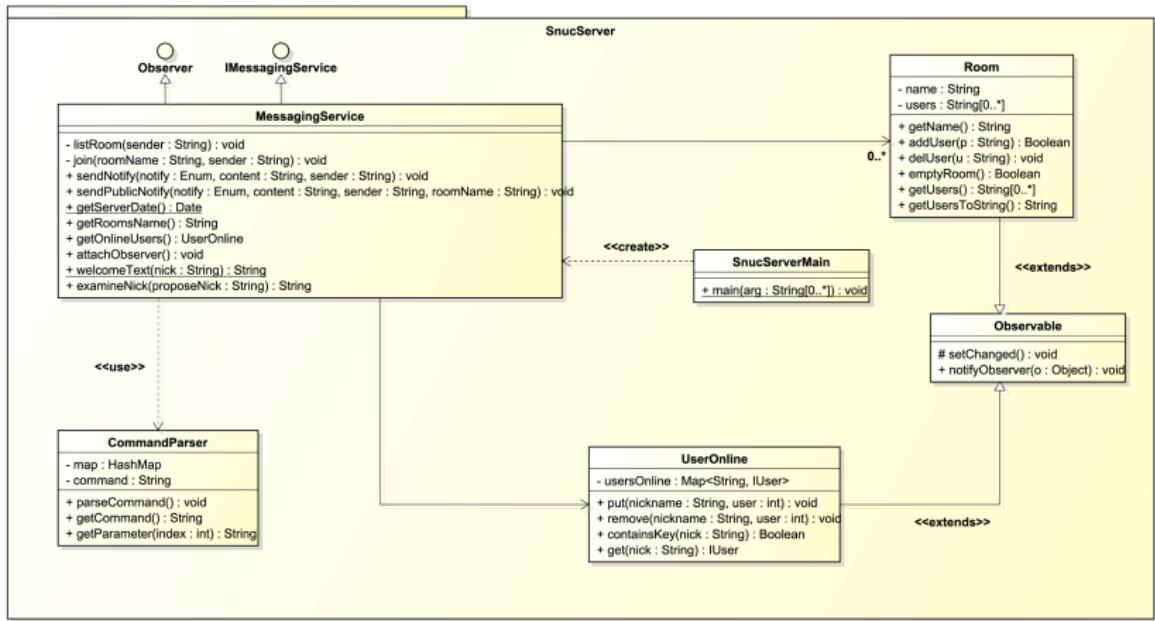


Figura 46 : DCD - Diagramma delle Classi: Package Snuc Server

Iterazione 2: Progettazione, UC3_SendPublicMessage - OP1

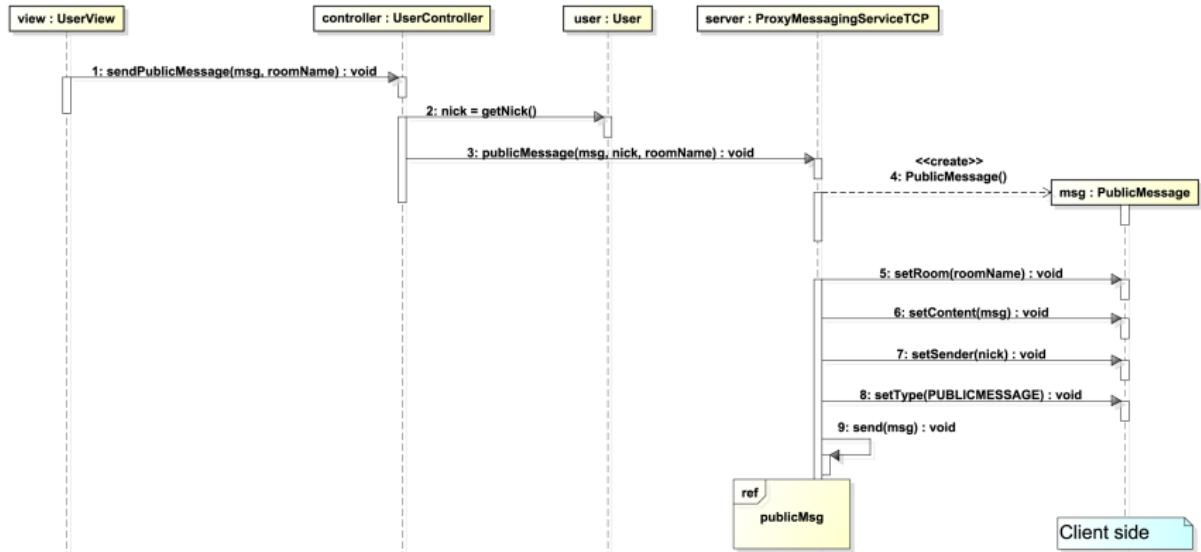


Figura 47 : SSD - OP1: sendMsg(msg) del modello di dominio (figura 42)

Iterazione 2: Progettazione, UC3_SendPublicMessage - OP2

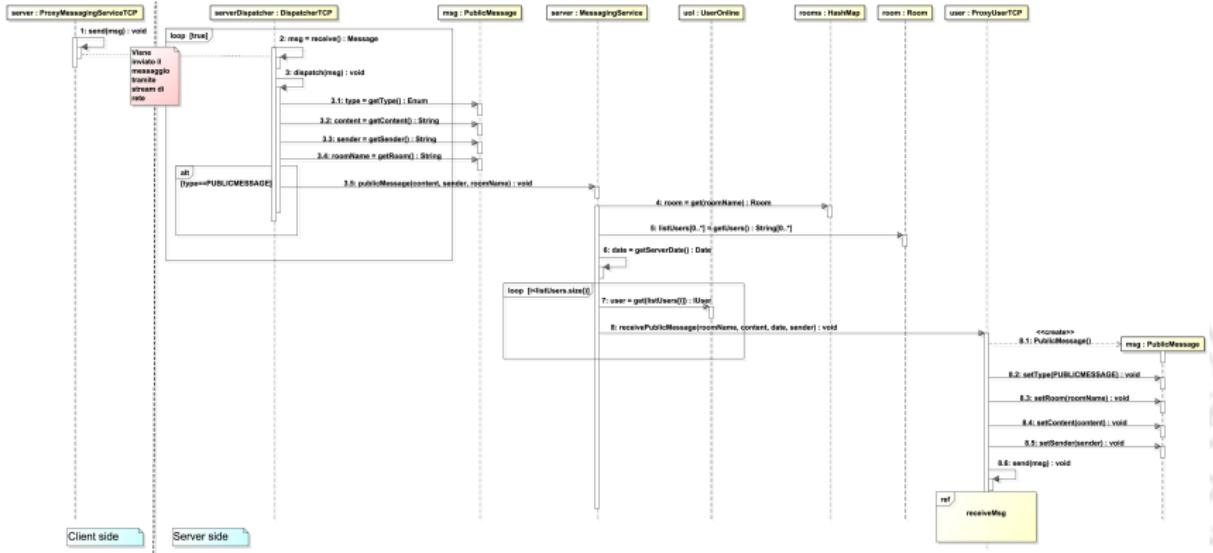


Figura 48 : SSD - OP2: pubblicMsg(msg) del modello di dominio (figura 42)

Iterazione 2: Progettazione, UC3_SendPublicMessage - OP3

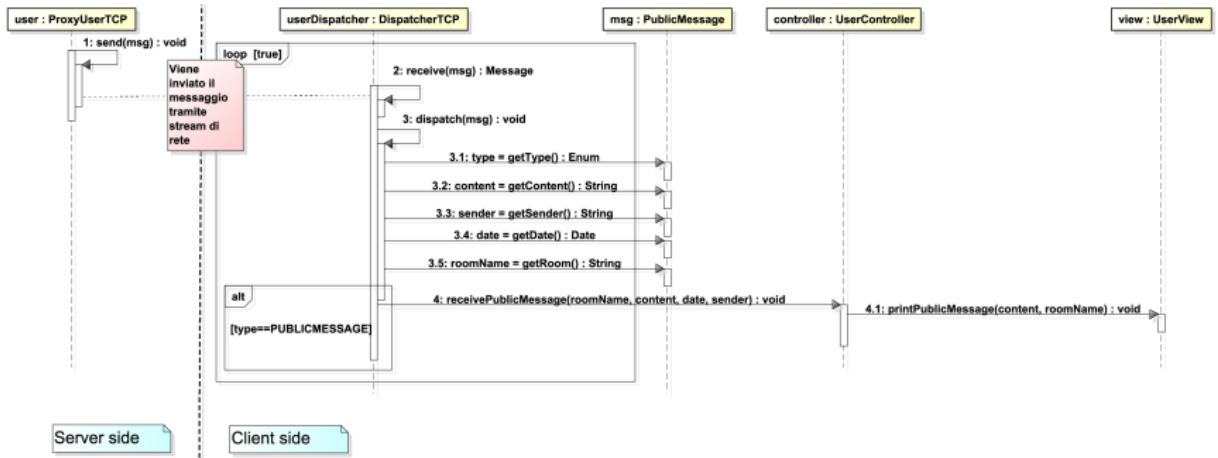


Figura 49 : SSD - OP3: receive(msg) del modello di dominio (figura 42)





Iterazione 2: Implementazione - UC3_SendPublicMessage

Gli screenshot di seguito rappresentano il risultato dell'implementazione effettuata per la seconda iterazione relativa al caso d'uso UC3_SendPublicMessage.

Il codice relativo delle iterazione_2 è reperibile nel repository GitHub del progetto.





Iterazione 2: Implementazione - UC3_SendPublicMessage

The screenshot shows the SNUC application interface. The title bar reads "SNUC". The menu bar includes "Snuc", "Server", and "Help". The main window is divided into three panels: "Rooms" on the left, "log" and "#Medical" tabs in the center, and "Users" on the right.

- Rooms Panel:** Lists available rooms: #Mathematics, #Pharmacy, **#Medical**, #ElectronicEngineering, and #ComputerScience. The "#Medical" room is currently selected.
- Log Tab:** Displays the following messages:
 - Sun Mar 08 16:13:29 CET 2015 Nicola has joined in #Medical
 - [16:18:12] Nicola>> Ciao Leo
 - [16:18:35] Leo>> Ciao Nicola
 - Sun Mar 08 16:22:34 CET 2015 Daniele has joined in #Medical
 - [16:23:18] Daniele>> Ehi ciao ragazzi..come va?
- Users Panel:** Lists the users in the current room: Leo, Nicola, and Daniele.
- Bottom Panel:** Contains a text input field and a "Send" button.

Figura 50 : UC3 - SendPublicMessage



Iterazione 3: Requisiti - UC4_SendPrivateMessage

Tabella 9 : Caso d'uso UC4_SendPrivateMessage

Nome caso d'uso	UC4_SendPrivateMessage
Portata	Applicazione Smart Network 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">① L'utente seleziona il destinatario del messaggio privato.② L'utente inserisce da tastiera il messaggio da inviare.③ Il messaggio viene inoltrato al destinatario selezionato.



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"> ▶ È possibile inviare messaggi confidenziali, autenticati e integri al server del servizio di messaggistica. ▶ 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 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. CONNECTION_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

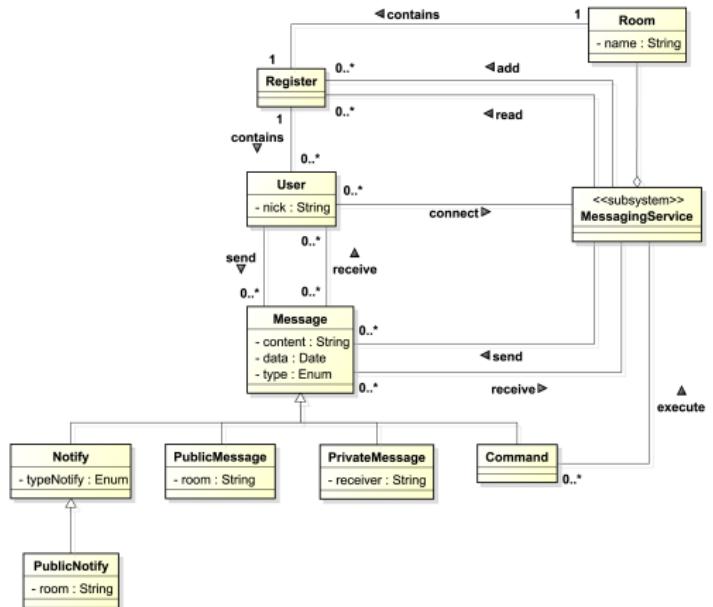


Figura 51 : UC3 - Modello di dominio

Iterazione 3: Analisi - UC4_SendPrivateMessage

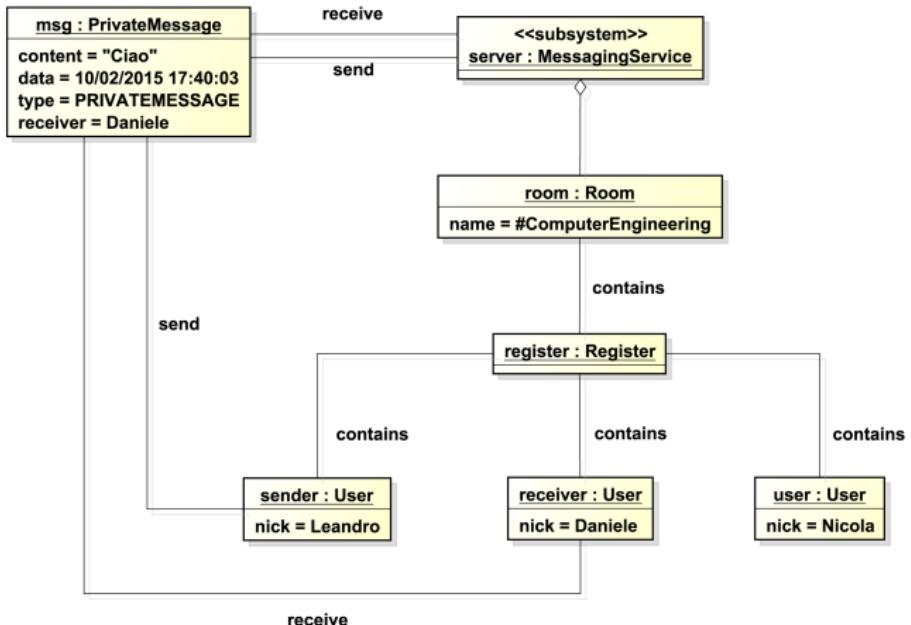


Figura 52 : UC3 - Oggetti di dominio

Iterazione 3: Analisi - UC4_SendPrivateMessage

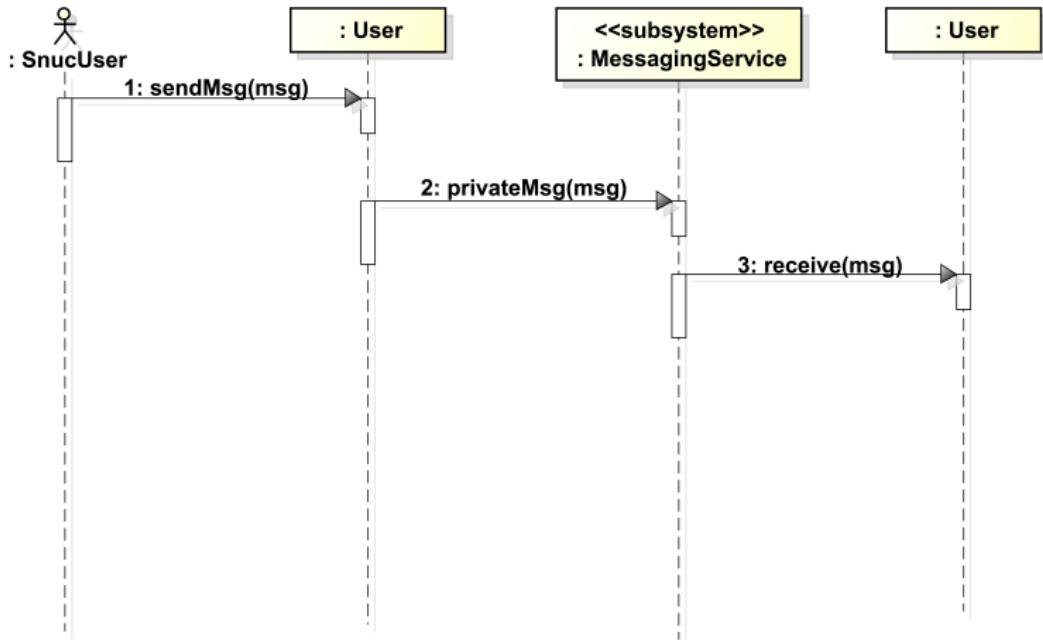


Figura 53 : UC4 - Diagramma di sequenza di sistema



Iterazione 3: Analisi, UC4 contratto CO6

Tabella 10 : UC3 Contratto CO6 - privateMsg

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



Iterazione 3: Progettazione - Descrizione

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: Progettazione Class Diagram Common UC4

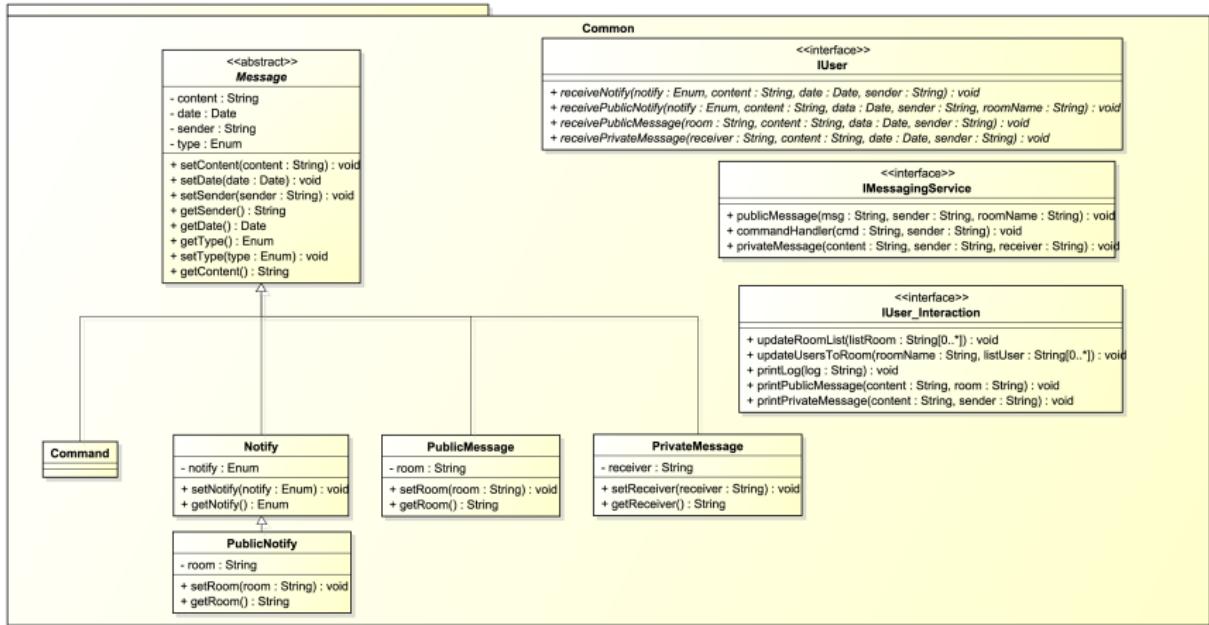


Figura 54 : DCD - Diagramma delle Classi: Package Common

Iterazione 3: Progettazione Class Diagram Snuc UC4

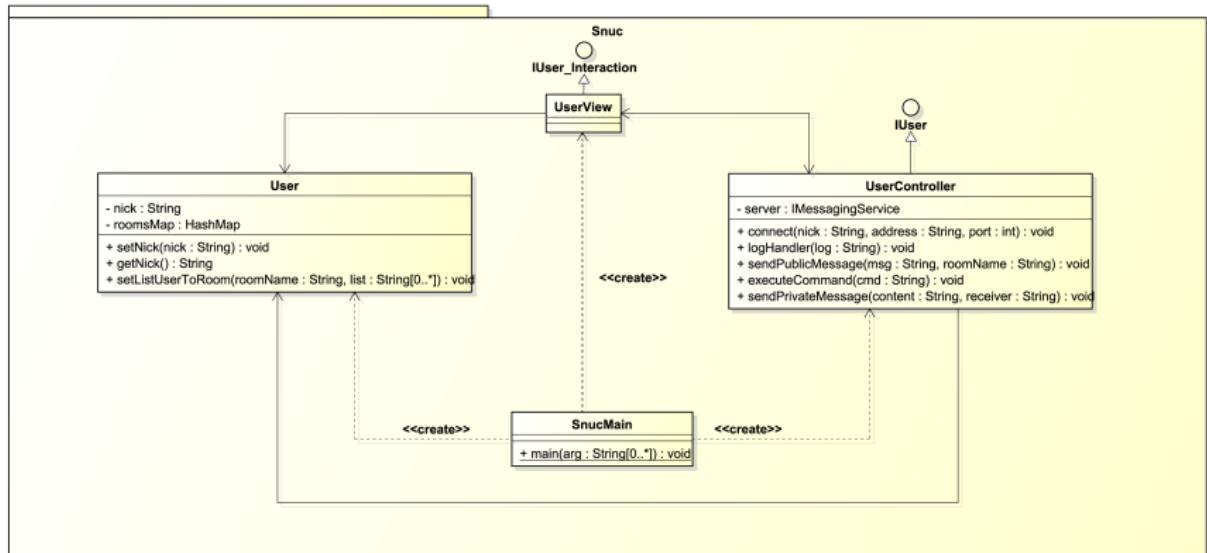


Figura 55 : DCD - Diagramma delle Classi: Package Snuc



Iterazione 3: Progettazione Class Diagram Connector UC4

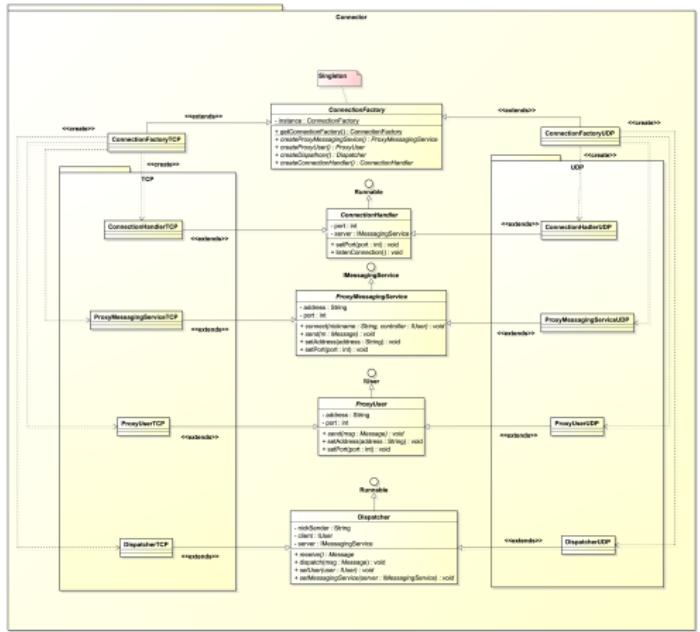


Figura 56 : DCD - Diagramma delle Classi: Package Connector

Iterazione 3: Progettazione Class Diagram Snuc Server UC4

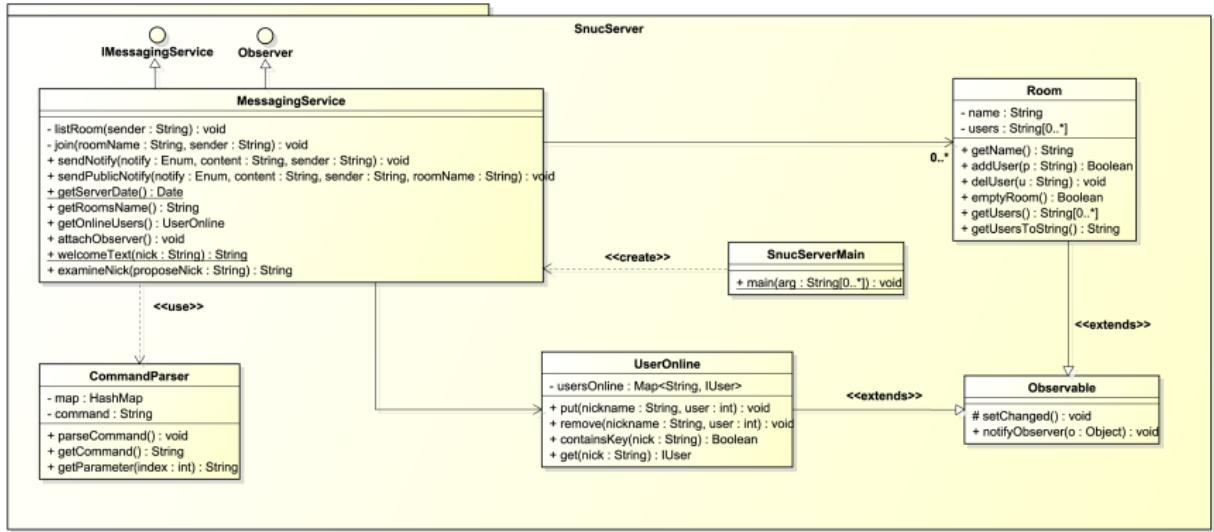


Figura 57 : DCD - Diagramma delle Classi: Package Snuc Server

Iterazione 3: Progettazione, UC4_SendPrivateMessage - OP1

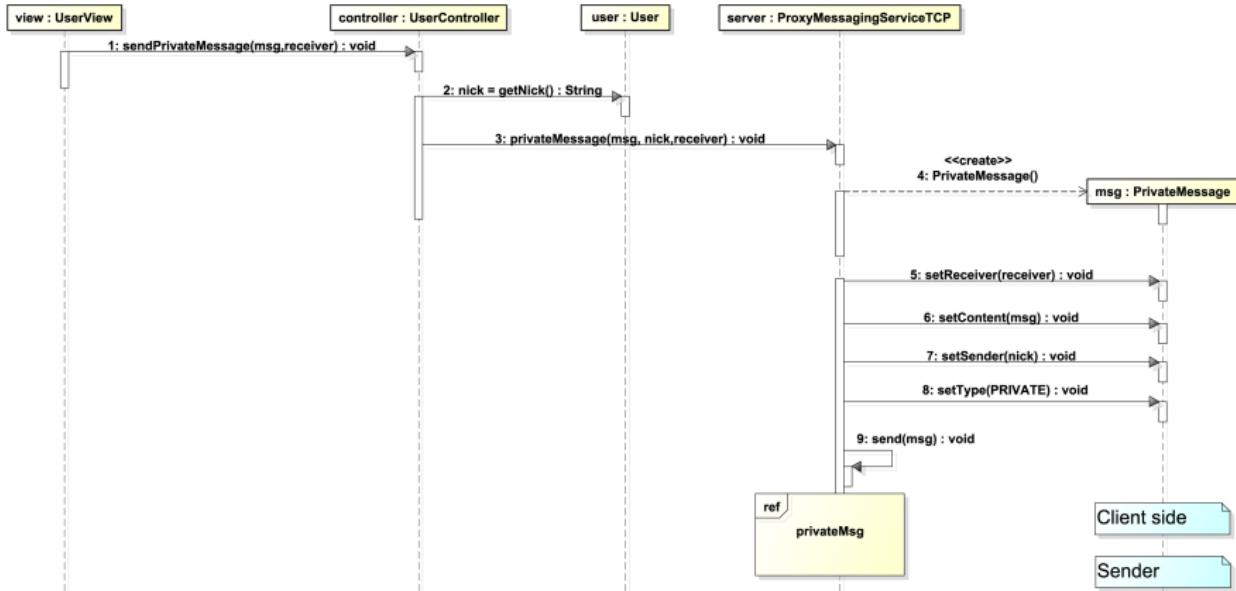


Figura 58 : SSD - OP1: sendMsg(msg) del modello di dominio (figura 53)

Iterazione 3: Progettazione, UC4_SendPrivateMessage - OP2

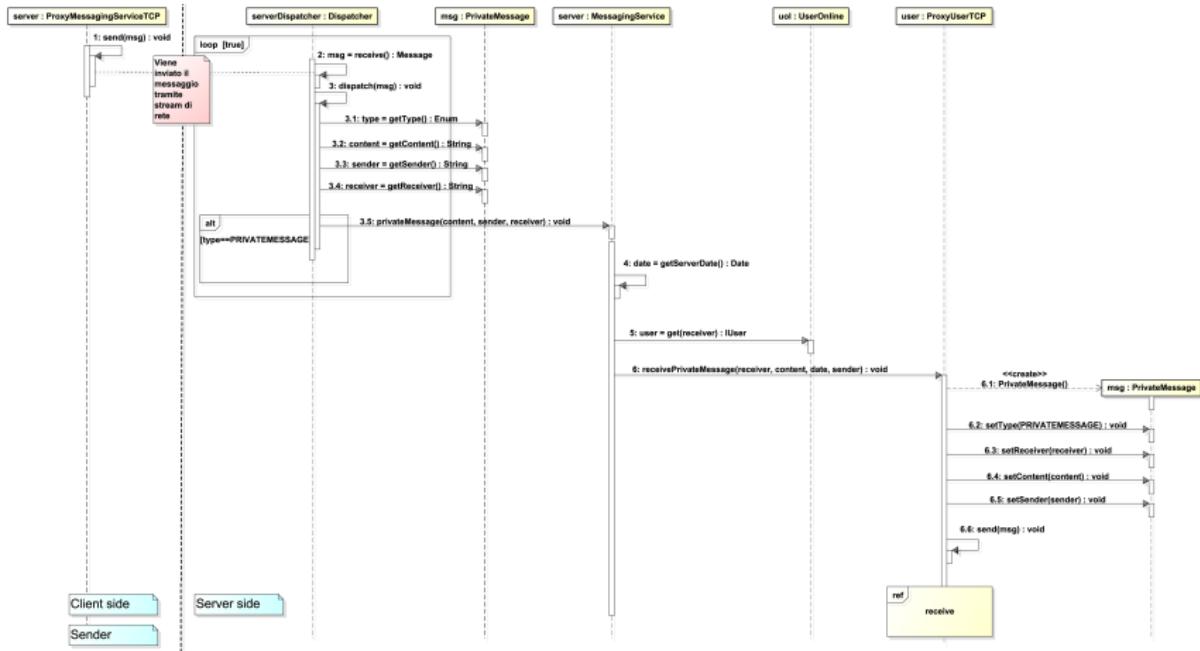


Figura 59 : SSD - OP2: privateMsg(msg) del modello di dominio (figura 53)

Iterazione 3: Progettazione, UC4_SendPrivateMessage - OP3

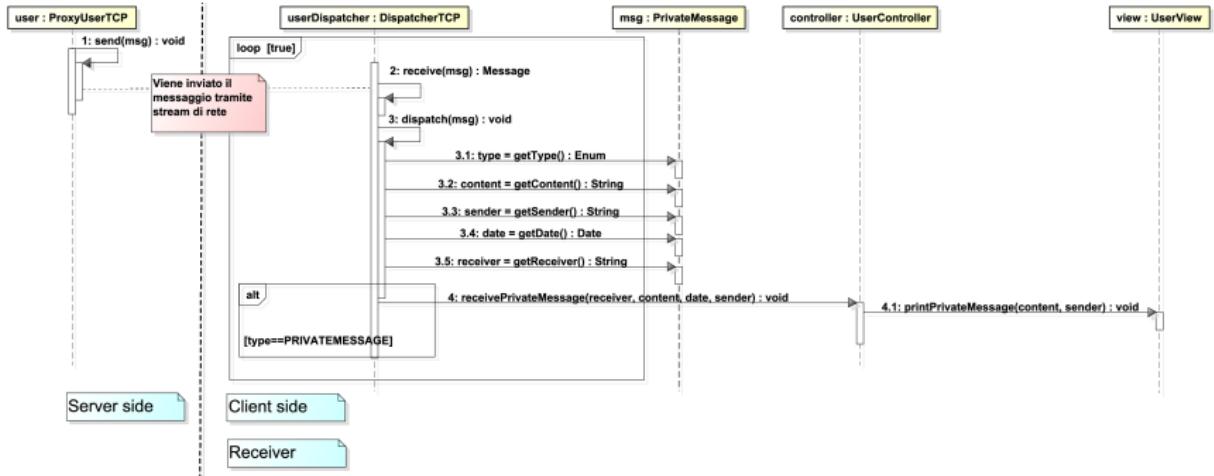


Figura 60 : SSD - OP3: receive(msg) del modello di dominio (figura 53)



Iterazione 3: Implementazione - UC4_SendPrivateMessage

Gli screenshot di seguito rappresentano il risultato dell'implementazione effettuata per la terza iterazione relativa al caso d'uso UC4 SendPrivateMessage.

Il codice relativo delle iterazione_3 è reperibile nel repository GitHub del progetto.





Iterazione 3: Implementazione - UC4_SendPrivateMessage

The screenshot shows the SNUC application window. The menu bar includes 'Snuc', 'Server', and 'Help'. The title bar says 'SNUC'. The left sidebar lists 'Rooms': #Mathematics, #Pharmacy, #Medical (selected), #ElectronicEngineering, and #ComputerScience. The main area has tabs for 'log', '#Medical' (selected), and 'Leo'. The '#Medical' tab displays a chat log:
[16:20:39] Leo>> puoi darmi il tuo email privata?
[16:21:25] Nicola>> nicola.didomenico@gmail.com

Figura 61 : UC4 - SendPrivateMessage



Conclusioni

Sviluppare questo progetto è stato un ottimo modo per imparare ed approfondire, mediante l'utilizzo diretto, le tecniche di analisi e progettazione di un software. Inoltre possiamo aggiungere che il lavoro di gruppo è stato formativo e ha reso la cosa molto simile ad un vero team di sviluppo. In conclusione speriamo di aver reso il più comprensibile possibile tutte le fasi affrontate.





Software Utilizzati

- Java SE 8 JDK
- JUnit 4.0 test framework Java
- NetBeans IDE 8.0.2
- Astah: UML and Modeling Tools
- L^AT_EX
- Git





Repository GitHub Progetto

- Documentazione e UML:

https://github.com/gnunicky/project_se_snuc_astah_latex.git

- Sorgenti Java delle varie iterazioni:

https://github.com/gnunicky/project_se_snuc_java.git





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>

