UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Ingegneria del Software

UniClass - Object Design Document

v1.3

DATA: 20/02/2026

Coordinatore del progetto:

Nome	Matricola
Giuseppe Sabetta	0512117895

Partecipanti:

Nome	Matricola
Giuseppe Sabetta	0512117895
Sara Gallo	0512117262
Saverio D'Avanzo	0512118330
Gerardo Antonio Cetrulo	0512117856

Scritto da:	Giuseppe Sabetta (GS), Sara Gallo (SG), Saverio D'Avanzo (SD), Gerardo Antonio Cetrulo (AC), Lucageneroso Cammarota (LC)

Revision History

Data	Versione	Descrizione	Autore
20/02/2026	1.3	Sostituzione del mapping a oggetti e refactoring dell'interface glossary Utente	LC
30/1/2025	1.2	Revisione del mapping e interfacce	GS, SG, SD, AC
17/12/2024	1.1	Correzione formattazione, cambio colori, miglioramento mapping, modifica interfacce	GS, SG, SD, AC
16/12/2024	1.0	Object Design Document	GS, SG, SD, AC

Indice

1 Introduzione

1.1 Object Design trade-offs

Ci sono varie considerazioni da seguire, dunque alcuni compromessi da valutare per raggiungere l'ottimalità del sistema.

I trade off sono i seguenti:

- **Flessibilità vs Stabilità:** La flessibilità è fondamentale per un sistema in crescita come UniClass. Una piattaforma destinata a studenti universitari deve essere in grado di adattarsi a esigenze mutevoli, come nuove funzionalità (es. integrazione di calendari personalizzati) o modifiche strutturali (es. cambi di orario). Tuttavia, l'adozione della flessibilità comporta un rischio maggiore di introdurre bug, dato che modifiche frequenti possono destabilizzare il sistema.

  Per bilanciare i rischi, si implementa:

  - **Testing:** per rilevare tempestivamente i bug
  - **Feature toggles:** abilitare/disabilitare funzionalità in base alla maturità del codice
  - **Versioning delle API:** garantire retrocompatibilità durante l'aggiunta di nuove funzionalità

- **Prestazioni vs Manutenibilità:** La manutenibilità è essenziale per un progetto con una lunga prospettiva di utilizzo e aggiornamento, come UniClass. Questo tipo di piattaforma deve essere facile da aggiornare per incorporare nuove richieste senza compromettere l'intero sistema.

  La manutenibilità però potrebbe comportare una leggera perdita di prestazioni.

  Per garantire comunque delle buone prestazioni, si procede:

  - **Utilizzando il paradigma MVC:** per migliorare la chiarezza del codice, riducendo il rischio di errori durante la manutenzione
  - **Refactoring regolare:** ottimizzando parti del codice critico senza compromettere la struttura generale

- **Costo vs Scalabilità:** Il sistema UniClass parte con soluzioni economiche e, in caso di traffico intenso e grande affluenza al servizio, UniClass verrà trasferito su una piattaforma cloud.

- **Personalizzazione vs Standardizzazione:** Il sistema UniClass è una piattaforma completamente personalizzabile e ad-hoc (per quanto riguarda le funzionalità introdotte nel RAD), dato che la stessa piattaforma deve essere in grado di permettere una navigabilità studiata in base alle preferenze didattiche dell'utente. Personalizzazioni troppo avanzate, però, possono aumentare la complessità del sistema, rendendo più difficili i test e il debug.

  Per non rendere il sistema troppo complesso, si procederà per:

  - **Personalizzazione modulare:** offrire opzioni preconfigurate e scalabili per evitare configurazioni eccessivamente granulari
  - **Feature preview:** introdurre gradualmente nuove opzioni di personalizzazione per testarne l'efficacia

- **Tempo di Esecuzione vs Memoria:** Ottimizzare per la memoria è una scelta pragmatica per un sistema con grandi quantità di dati (es. orari, prenotazioni, profili). Una buona gestione della memoria riduce i costi operativi e migliora la stabilità complessiva. Tuttavia, tempi di esecuzione più lunghi potrebbero compromettere l'esperienza utente.

  Per rendere il sistema comunque efficiente si utilizzerà:

  - **Database ottimizzato:** utilizzando indici e partizionamento per migliorare le query senza impatti negativi sulla memoria

TRADE OFF	
Flessibilità	Stabilità
Manutenibilità	Prestazioni
Scalabilità	Costo
Personalizzazione	Standardizzazione
Memoria	Tempo di Esecuzione

Tabella 2: Trade-off di Object Design

1.2 Interface Document Guidelines

In questa sezione avremo un insieme di regole da utilizzare nella progettazione delle interfacce:

- **Principio di Segregazione delle Interfacce:** Assicurarsi che per il cliente non ci siano metodi non necessari. Ogni interfaccia deve essere specifica e aderire a un solo scopo.
- **Le trasformazioni devono essere effettuate in isolamento:** (da modificare, per ogni trasformazione mettere un trattino)
- **Gestione delle eccezioni:**
  - Per ogni funzionalità, catturare le eccezioni previste o crearne di nuove coerenti con l'errore.
  - Stampare l'eccezione e importarla in un log file.
  - Evitare l'uso generico di `catch (Exception e)` senza gestione dettagliata.
- **Validazione dati input:** controllare valori nulli, formati errati o valori fuori range per metodi dove il dominio degli input è variabile/dinamico
- **Principio di Liskov:** Seguire un'ereditarietà rigorosa. Le sottoclassi devono essere sostituibili alle super-classi senza modificare il codice client. Nessun metodo deve violare le aspettative degli sviluppatori/client.

- **Parentesi graffe {}:** dopo l'inizio di un metodo e finire un rigo dopo l'ultima riga di codice
- **Parametri nei metodi:**
  - In presenza di più parametri per un metodo, inserire (parametro1, parametro2, ...), mettendo la virgola subito dopo il parametro e uno spazio per l'eventuale parametro successivo
  - Usare nomi generici durante la scrittura iniziale per semplificare la documentazione
- **Commenti nei metodi:**
  - Utilizzare `//` per spiegare istruzioni complesse o di difficile comprensione.
  - Aggiungere commenti multilinea `/* */` all'inizio di ogni classe con una spiegazione dettagliata di scopo e responsabilità
- **Gestione della spaziatura:**
  - Aumentare la leggibilità con un'adeguata gestione degli spazi
  - Aggiungere una riga vuota tra metodi, blocchi logici e all'interno di costrutti lunghi
- **Lunghezza delle righe:** Limitare ogni riga a 80-120 caratteri per facilitare la lettura del codice, soprattutto su schermi piccoli o IDE con più finestre aperte.
- **Metodi chiari e concisi:** Ogni metodo deve avere una sola responsabilità. Non devono essere presenti metodi monolitici con più scopi.
- **Nomi Descrittivi:**
  - Usare nomi di metodi e classi auto-esplicativi, come `calcolaOrario()`, etc.
  - Evitare acronimi o abbreviazioni non comprensibili facilmente.
- **Documentazione delle interfacce:**
  - Ogni interfaccia deve avere una breve descrizione del suo scopo, dei suoi metodi e di cosa ritornano. Bisogna usare **Javadoc**.
  - Le interfacce non devono dipendere da classi concrete. Utilizzare tipi astratti o generici dove possibile.
  - Inserire solo costanti (`final static`) all'interno delle interfacce, evitando implementazioni dirette, per fini di sicurezza e usabilità.
- **Indentazione Standard:** usare una tabulazione uniforme per indentare il codice. Non sono permessi spazi per l'indentazione.
- **Nomenclatura uniforme:**
  - Classi e Interfacce: PascalCase
  - Metodi: camelCase
  - Costanti: UPPERCASE
- **Gestione del dead code:** non sono permesse implementazioni di metodi non utilizzati o non commentati

1.3 Design Patterns

Nello sviluppo del sistema **UniClass**, l'adozione dei **design patterns** si rivela essenziale per affrontare in maniera efficace i compromessi individuati durante la progettazione del sistema. I design patterns forniscono soluzioni collaudate a problemi ricorrenti, permettendo di migliorare **manutenibilità, scalabilità** e **flessibilità** del codice, senza introdurre complessità eccessiva. In un contesto caratterizzato da esigenze contrastanti, come **flessibilità vs stabilità** o **prestazioni vs manutenibilità**, l'uso dei pattern garantisce una struttura chiara e robusta, facilitando l'integrazione di nuove funzionalità e la gestione dei cambiamenti. Attraverso approcci standardizzati e modulari, come l'implementazione del paradigma **MVC** e la personalizzazione modulare, possiamo mantenere un equilibrio ottimale tra personalizzazione e stabilità del sistema, migliorando al contempo l'esperienza di sviluppo e utilizzo della piattaforma.

Pattern architetturali:

- **Service Pattern:** Il Service Pattern è un pattern architetturale che separa la logica di business dall'accesso ai dati, migliorando la manutenibilità e l'organizzazione del codice. In UniClass, il Service Pattern viene utilizzato per:
  - Separare la logica di business dai repository: i servizi interagiscono con i repository senza esporre direttamente la logica di accesso ai dati.
  - Facilitare il testing: i servizi possono essere testati separatamente simulando i repository.
  - Migliorare la scalabilità: ogni servizio può essere esteso o modificato senza impattare direttamente i controller o i repository.
  - L'adozione di questo pattern in UniClass garantisce una separazione chiara delle responsabilità e facilita l'integrazione di nuove funzionalità senza modificare il core del sistema.
- **Repository Pattern:** Il Repository Pattern è un pattern architetturale che fornisce un livello di astrazione tra il livello di accesso ai dati e la logica di business, migliorando la manutenibilità e testibilità del codice. In UniClass, il Repository Pattern viene utilizzato per:
  - Centralizzare l'accesso ai dati, effettuando le operazioni CRUD sul database, riducendo di molto il codice.
  - Facilitare la sostituzione del database, dato che l'accesso ai dati è incapsulato nel repository, permettendo di cambiare database senza modificare il resto dell'applicazione
  - Migliorare la testibilità, dato che i repository possono essere facilmente sostituiti con mock nei test.

1.4 Definition, acronyms and abbreviations

In questo documento, sono state utilizzate diverse abbreviazioni e termini che richiedono la specifica della definizione per aumentare la comprensione del documento.

Acronimo/Abbreviazione	**Definizione**
MVC	Paradigma di progettazione software che separa le componenti di Model, View e Control
Javadoc	Strumento di documentazione incluso nel JDK Java che genera html per descrizione delle specifiche di classi e metodi
JDK	Ambiente di sviluppo di applicazioni in linguaggio Java
DAO	Pattern progettuale per astrarre e incapsulare l'accesso ai dati di un'applicazione, fornendo interfacce per operazioni CRUD
CRUD	Rappresentazione delle quattro operazioni fondamentali sui Database, Create, Retrieve, Update, Delete

1.5 Mapping a Oggetti

Il sistema UniClass adotta un'architettura basata sul principio di **Composizione** piuttosto che su una rigida gerarchia di ereditarietà. L'entità centrale è `Utente`, che rappresenta l'identità digitale univoca. Le informazioni specifiche accademiche sono gestite tramite una relazione di composizione con l'entità `Accademico` (relazione 1-a-1 opzionale), mentre i permessi e le funzionalità sono determinati dai `Ruoli` assegnati all'utente. Il controllo degli accessi e la logica di business centralizzata sono delegati al componente `UserDirectory`, che funge da Facade per tutti i servizi legati agli utenti.

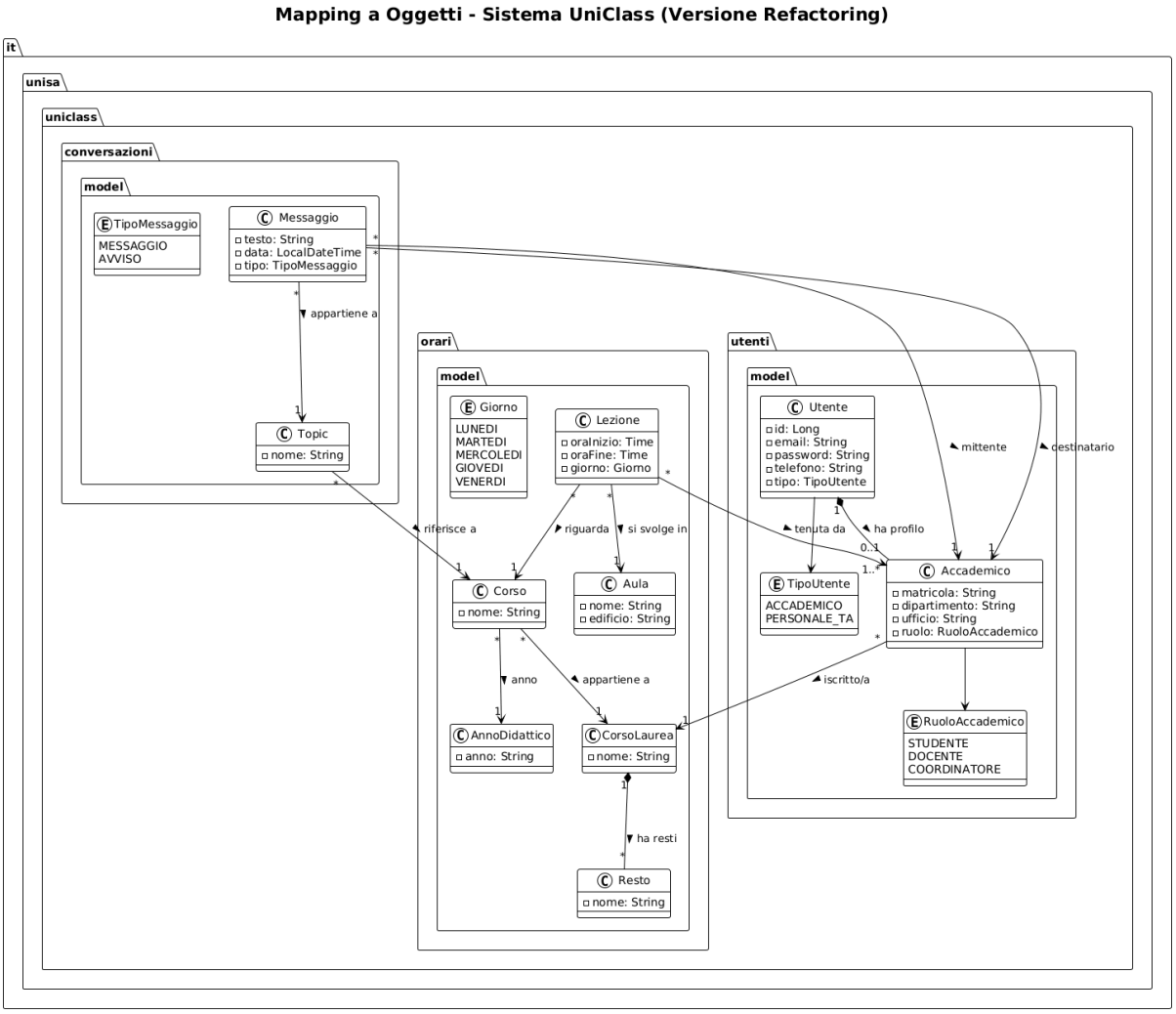

Figura 1: Mapping a Oggetti

1.6 References

- RequirementsAnalysisDocument_UniClass.pdf
- SystemDesignDocument_UniClass.pdf

- Object Oriented Software Engineering using UML, Patterns and Java Third Edition – Bruegge, Dutoit

2 Packages

Il sistema UniClass presenta un packaging basato sul layering e partitioning visibile dall'SDD, seguendo il paradigma MVC, dividendo ogni sottosistema in model, view e control. Il model rappresenta l'oggetto in sé, i services rappresentano i DAO per la comunicazione tra i model e le repository persistenti e il controller per le servlet. I sottosistemi sono orari, utenti, conversazioni, notifiche, esami, mentre un ultimo package importante sarà common, contenente utility (come design patterns o funzioni ausiliarie), exception (eccezioni ad-hoc), config (file di configurazione dell'ambiente), security (per controlli e filtri presenti sull'ambiente) e controller, per servlet comuni.

Packages:

```
it.unisa.uniclass
├── orari
│   ├── model
│   ├── service
│   │   └── dao
│   └── controller
├── utenti
│   ├── model
│   ├── service
│   │   └── dao
│   └── controller
├── conversazioni
│   ├── model
│   ├── service
│   │   └── dao
│   └── controller
└── common
    ├── utils
    ├── exceptions
    ├── config.database
    ├── Filter
    └── security
```

Package it.unisa.uniclass.utenti:

- **model**: Contiene le entità dominio (`Utente`, `Accademico`, `Ruolo`, `Tipo`).
- **service**: Contiene il Service Layer. Il componente principale è `UserDirectory` (Facade) che coordina l'accesso agli utenti. Sono stati rimossi i servizi specifici legacy (`StudenteService`, `DocenteService`, ecc.) per ridurre l'accoppiamento.
- **service.dao**: Contiene i Data Access Object (`UtenteDAO`, `AccademicoDAO`) per la persistenza.

- **controller**: Contiene le Servlet per la gestione delle richieste HTTP (es. `LoginServlet`, `AttivaUtentiServlet`).

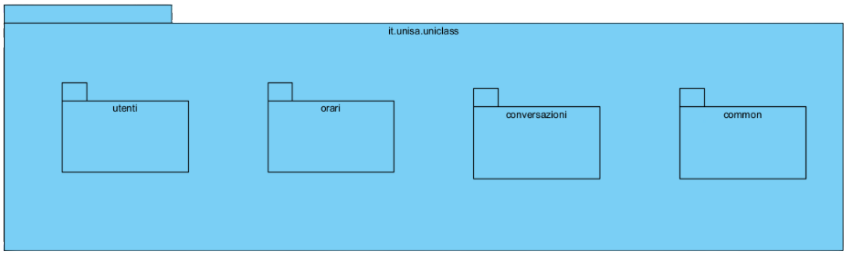

Figura 2: Package it.unisa.uniclass

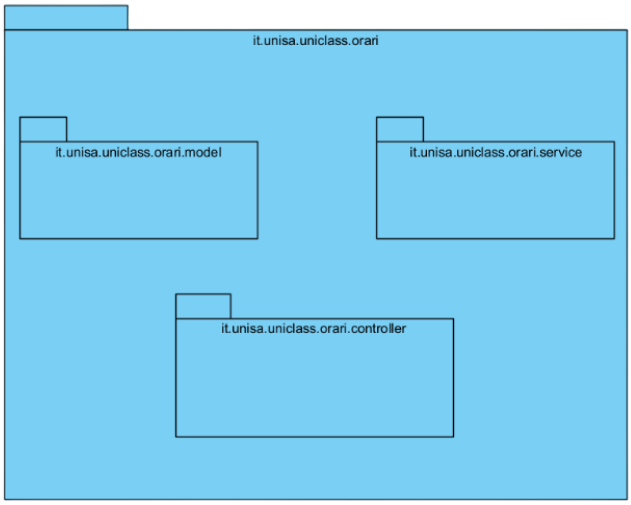

Figura 3: Package it.unisa.uniclass.orari

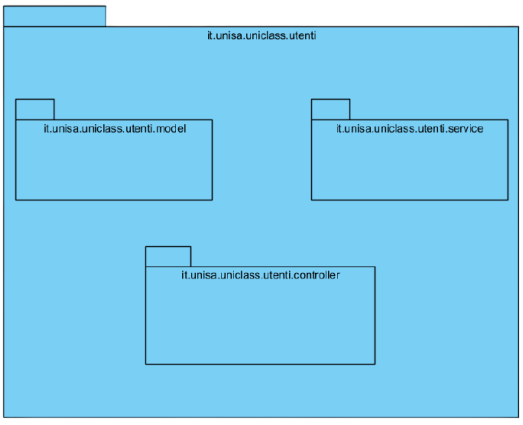

Figura 4: Package it.unisa.uniclass.utenti

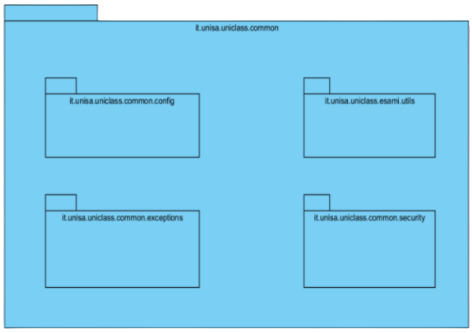

Figura 5: Package it.unisa.uniclass.common

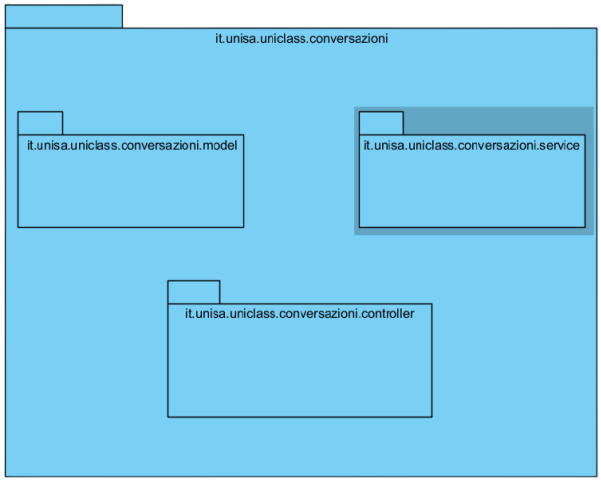

Figura 6: Package it.unisa.uniclass.conversazioni

3 Class Interfaces Glossary

Javadoc è uno strumento di documentazione fornito con il JDK (Java Development Kit) che consente di generare documentazione in formato HTML direttamente dal codice sorgente Java. Funziona analizzando i commenti strutturati (detti anche "doc comments") inseriti sopra classi, metodi, campi e costruttori.

Questi commenti sono delimitati da `/** ... */` e seguono una sintassi specifica che include tag come `@param`, `@return`, `@throws`, etc.

Per ogni package sarà possibile avere un'**interfaccia** con i metodi presenti nel repository pattern in questione.

3.1 Package Gestione Orari

Nome Interfaccia	AnnoDidatticoRemote
Scopo	Interfaccia relativa al servizio della gestione degli anni didattici di un corso di laurea.
Invariante di classe	

Nome Metodo	+ trovaAnno(String anno) : AnnoDidattico
Descrizione Metodo	Il metodo permette di trovare un certo annoDidattico in base al parametro.
Pre-condizioni	context AnnoDidatticoRemote::trovaId(String) : AnnoDidattico
pre: anno $\neq$ null and anno.size() $>$ 0	
Post-condizioni	context AnnoDidatticoRemote::trovaId(String) : AnnoDidattico
post: result$\rightarrow$forAll(a | a.anno = anno)	

Nome Metodo	+ trovaAnno(int id) : AnnoDidattico
Descrizione Metodo	Il metodo permette di trovare un AnnoDidattico dato il suo identificativo.
Pre-condizioni	
Post-condizioni	context AnnoDidatticoRemote::trovaId(int) : AnnoDidattico
post: result.id = id	

Nome Metodo	+ trovaTutti() : List(AnnoDidattico)
Descrizione Metodo	Il metodo restituisce la lista di tutti gli anni didattici disponibili.
Pre-condizioni	
Post-condizioni	context AnnoDidatticoRemote::trovaTutti() : List(AnnoDidattico)
post: result$\rightarrow$size() $\geq$ 0	

Nome Metodo	+ trovaTuttiCorsoLaurea(long id) : List(AnnoDidattico)
Descrizione Metodo	Il metodo restituisce tutti gli anni didattici associati a un corso di laurea specifico.
Pre-condizioni	context AnnoDidatticoRemote::trovaTuttiCorsoLaurea(long) : List(AnnoDidattico)
pre: id > 0	
Post-condizioni	context AnnoDidatticoRemote::trovaTuttiCorsoLaurea(long) : List(AnnoDidattico)
post: result→forAll(a | a.corsoLaurea.id = id)	

Nome Metodo	+ trovaCorsoLaureaNome(long id, String anno) : AnnoDidattico
Descrizione Metodo	Il metodo restituisce un AnnoDidattico dato l'ID del corso di laurea e l'anno.
Pre-condizioni	context AnnoDidatticoRemote::trovaCorsoLaureaNome(long, String) : AnnoDidattico
pre: id > 0 and anno ≠ null and anno.size() > 0	
Post-condizioni	context AnnoDidatticoRemote::trovaCorsoLaureaNome(long, String) : AnnoDidattico
post: result.corsoLaurea.id = id and result.anno = anno	

Nome Metodo	+ aggiungiAnno(AnnoDidattico annoDidattico) : void
Descrizione Metodo	Il metodo permette di aggiungere un nuovo AnnoDidattico.
Pre-condizioni	context AnnoDidatticoRemote::aggiungiAnno(AnnoDidattico)
pre: annoDidattico ≠ null	
Post-condizioni	context AnnoDidatticoRemote::aggiungiAnno(AnnoDidattico)
post: AnnoDidatticoRemote::trovaTutti()→includes(annoDidattico)	

Nome Metodo	+ rimuoviAnno(AnnoDidattico annoDidattico) : void
Descrizione Metodo	Il metodo permette di rimuovere un AnnoDidattico esistente.
Pre-condizioni	context AnnoDidatticoRemote::rimuoviAnno(AnnoDidattico)
pre: annoDidattico ≠ null and AnnoDidatticoRemote::trovaTutti()→includes(annoDidattico)	
Post-condizioni	context AnnoDidatticoRemote::rimuoviAnno(AnnoDidattico)
post: not AnnoDidatticoRemote::trovaTutti()→includes(annoDidattico)	

Nome Interfaccia	AulaRemote
Scopo	Interfaccia relativa al servizio della gestione delle aule in un ateneo.
Invariante di classe	context Aula inv: AulaDAO::trovaTutte()→forAll(a1, a2 | a1 ≠ a2 implies a1.nome ≠ a2.nome)
and AulaDAO::trovaTutte()→forAll(a | a.edificio ≠ null and a.edificio.size() > 0)	

Nome Metodo	+ trovaAula(int id) : Aula
Descrizione Metodo	Il metodo trova un'aula in base all'identificativo nel parametro.
Pre-condizioni	context AulaDAO::trovaAula(int) : Aula
pre: id > 0	
Post-condizioni	context AulaDAO::trovaAula(int) : Aula
post: result.id = id	

Nome Metodo	+ trovaAula(String nome) : Aula
Descrizione Metodo	Il metodo trova un'aula in base al nome nel parametro.
Pre-condizioni	context AulaDAO::trovaAula(String) : Aula
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context AulaDAO::trovaAula(String) : Aula
post: result.nome = nome	

Nome Metodo	+ trovaTutte() : List(Aula)
Descrizione Metodo	Il metodo trova tutte le aule esistenti.
Pre-condizioni	
Post-condizioni	context AulaDAO::trovaTutte() : List(Aula)
post: result→size() ≥ 0	

Nome Metodo	+ trovaAuleEdificio(String edificio) : List(Aula)
Descrizione Metodo	Il metodo trova tutte le aule di uno specifico edificio.
Pre-condizioni	context AulaDAO::trovaAuleEdificio(String) : List(Aula)
pre: edificio ≠ null and edificio.size() > 0	
Post-condizioni	context AulaDAO::trovaAuleEdificio(String) : List(Aula)
post: result→forAll(a | a.edificio = edificio)	

Nome Metodo	+ trovaEdifici() : List(String)
Descrizione Metodo	Il metodo trova tutti gli edifici esistenti.
Pre-condizioni	
Post-condizioni	context AulaDAO::trovaEdifici() : List(String)
post: result→size() ≥ 0	

Nome Metodo	+ aggiungiAula(Aula aula) : void
Descrizione Metodo	Il metodo aggiunge un'aula.
Pre-condizioni	context AulaDAO::aggiungiAula(Aula)
pre: aula ≠ null	
Post-condizioni	context AulaDAO::aggiungiAula(Aula)
post: AulaDAO::trovaTutte()→includes(aula)	

Nome Metodo	+ rimuoviAula(Aula aula) : void
Descrizione Metodo	Il metodo rimuove un'aula.
Pre-condizioni	context AulaDAO::rimuoviAula(Aula)
pre: aula ≠ null and AulaDAO::trovaTutte()→includes(aula)	
Post-condizioni	context AulaDAO::rimuoviAula(Aula)
post: not AulaDAO::trovaTutte()→includes(aula)	

Nome Interfaccia	CorsoRemote
Scopo	Interfaccia relativa al servizio della gestione dei corsi dei corsi di Laurea.
Invariante di classe	context Corso inv: CorsoDAO::trovaTutti()→forAll(c1, c2 | c1 ≠ c2 implies c1.nome ≠ c2.nome)
and CorsoDAO::trovaTutti()→forAll(c | c.corsoLaurea ≠ null)	

Nome Metodo	+ trovaCorso(long id) : Corso
Descrizione Metodo	Questo metodo trova un Corso utilizzando il suo ID.
Pre-condizioni	context CorsoDAO::trovaCorso(long) : Corso
pre: id > 0	
Post-condizioni	context CorsoDAO::trovaCorso(long) : Corso
post: result.id = id	

Nome Metodo	+ trovaCorsiCorsoLaurea(String nomeCorsoLaurea) : List(Corso)
Descrizione Metodo	Questo metodo trova tutti i corsi associati a un determinato corso di laurea.
Pre-condizioni	context CorsoDAO::trovaCorsiCorsoLaurea(String) : List(Corso)
pre: nomeCorsoLaurea ≠ null and nomeCorsoLaurea.size() > 0	
Post-condizioni	context CorsoDAO::trovaCorsiCorsoLaurea(String) : List(Corso)
post: result→forAll(c | c.corsoLaurea.nome = nomeCorsoLaurea)	

Nome Metodo	+ trovaTutti() : List(Corso)
Descrizione Metodo	Questo metodo recupera tutti i corsi esistenti.
Pre-condizioni	
Post-condizioni	context CorsoDAO::trovaTutti() : List(Corso)
post: result→size() ≥ 0	

Nome Metodo	+ aggiungiCorso(Corso corso) : void
Descrizione Metodo	Questo metodo aggiunge o aggiorna un Corso nel database.
Pre-condizioni	context CorsoDAO::aggiungiCorso(Corso) : void
pre: corso ≠ null	
Post-condizioni	context CorsoDAO::aggiungiCorso(Corso) : void
post: CorsoDAO::trovaTutti()→includes(corso)	

Nome Metodo	+ rimuoviCorso(Corso corso) : void
Descrizione Metodo	Questo metodo rimuove un corso esistente.
Pre-condizioni	context CorsoDAO::rimuoviCorso(Corso) : void
pre: corso ≠ null and CorsoDAO::trovaTutti()→includes(corso)	
Post-condizioni	context CorsoDAO::rimuoviCorso(Corso) : void
post: not CorsoDAO::trovaTutti()→includes(corso)	

Nome Interfaccia	CorsoLaureaDAO
Scopo	Interfaccia relativa alla gestione dei corsi di Laurea esistenti.
Invariante di classe	context CorsoLaurea inv: CorsoLaureaDAO::trovaTutti()→forAll(c1, c2 | c1 ≠ c2 implies c1.nome ≠ c2.nome)

Nome Metodo	+ trovaCorsoLaurea(long id) : CorsoLaurea
Descrizione Metodo	Questo metodo trova un corso laurea esistente.
Pre-condizioni	context CorsoLaureaDAO::trovaCorsoLaurea(long) : CorsoLaurea
pre: id > 0	
Post-condizioni	context CorsoLaureaDAO::trovaCorsoLaurea(long) : CorsoLaurea
post: result.id = id	

Nome Metodo	+ trovaCorsoLaurea(String nome) : CorsoLaurea
Descrizione Metodo	Questo metodo trova un corso laurea esistente.
Pre-condizioni	context CorsoLaureaDAO::trovaCorsoLaurea(String) : CorsoLaurea
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context CorsoLaureaDAO::trovaCorsoLaurea(String) : CorsoLaurea
post: result.nome = nome	

Nome Metodo	+ trovaTutti() : List(CorsoLaurea)
Descrizione Metodo	Questo metodo recupera tutti i corsi di laurea presenti nel database.
Pre-condizioni	
Post-condizioni	context CorsoLaureaDAO::trovaTutti() : List(CorsoLaurea)
post: result→size() ≥ 0	

Nome Metodo	+ aggiungiCorsoLaurea(CorsoLaurea corsoLaurea) : void
Descrizione Metodo	Questo metodo aggiunge o aggiorna un CorsoLaurea nel database.
Pre-condizioni	context CorsoLaurea-DAO::aggiungiCorsoLaurea(CorsoLaurea) : void
pre: corsoLaurea ≠ null	
Post-condizioni	context CorsoLaurea-DAO::aggiungiCorsoLaurea(CorsoLaurea) : void
post: CorsoLaureaDAO::trovaTutti()→includes(corsoLaurea)	

Nome Metodo	+ rimuoviCorsoLaurea(CorsoLaurea corsoLaurea) : void
Descrizione Metodo	Questo metodo rimuove un CorsoLaurea dal database.
Pre-condizioni	context CorsoLaurea-DAO::rimuoviCorsoLaurea(CorsoLaurea) : void
pre: corsoLaurea ≠ null and CorsoLaureaDAO::trovaTutti()→includes(corsoLaurea)	
Post-condizioni	context CorsoLaurea-DAO::rimuoviCorsoLaurea(CorsoLaurea) : void
post: not CorsoLaureaDAO::trovaTutti()→includes(corsoLaurea)	

Nome Interfaccia	LezioneRemote
Scopo	Interfaccia relativa al servizio della gestione delle lezioni esistenti.
Invariante di classe	context Lezione inv: self.oraInizio < self.oraFine
context Lezione inv: LezioneDAO::trovaTutte()→forAll(l1, l2 | l1 ≠ l2 implies (l1.aula = l2.aula implies not (l1.oraInizio < l2.oraFine and l1.oraFine > l2.oraInizio)))	

Nome Metodo	+ trovaLezione(long id) : Lezione
Descrizione Metodo	Questo metodo trova una Lezione utilizzando il suo ID.
Pre-condizioni	context LezioneDAO::trovaLezione(long) : Lezione
pre: id > 0	
Post-condizioni	context LezioneDAO::trovaLezione(long) : Lezione
post: result.id = id	

Nome Metodo	+ trovaLezioniCorso(String nomeCorso) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni associate a un determinato corso.
Pre-condizioni	context LezioneDAO::trovaLezioniCorso(String) : List(Lezione)
pre: nomeCorso ≠ null and nomeCorso.size() > 0	
Post-condizioni	context LezioneDAO::trovaLezioniCorso(String) : List(Lezione)
post: result→forAll(l | l.corso.nome = nomeCorso)	

Nome Metodo	+ trovaLezioniOre(Time oraInizio, Time oraFine) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni in una determinata fascia oraria.
Pre-condizioni	context LezioneDAO::trovaLezioniOre(Time, Time) : List(Lezione)
pre: oraInizio < oraFine	
Post-condizioni	context LezioneDAO::trovaLezioniOre(Time, Time) : List(Lezione)
post: result→forAll(l | l.oraInizio ≥ oraInizio and l.oraFine ≤ oraFine)	

Nome Metodo	+ trovaLezioniOreGiorno(Time oraInizio, Time oraFine, Giorno giorno) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni in una determinata fascia oraria e giorno della settimana.
Pre-condizioni	context LezioneDAO::trovaLezioniOreGiorno(Time, Time, Giorno) : List(Lezione)
pre: oraInizio < oraFine and giorno ≠ null	
Post-condizioni	context LezioneDAO::trovaLezioniOreGiorno(Time, Time, Giorno) : List(Lezione)
post: result→forAll(l | l.oraInizio ≥ oraInizio and l.oraFine ≤ oraFine and l.giorno = giorno)	

Nome Metodo	+ trovaLezioniAule(String nome) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni in un'aula specifica.
Pre-condizioni	context LezioneDAO::trovaLezioniAule(String) : List(Lezione)
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context LezioneDAO::trovaLezioniAule(String) : List(Lezione)
post: result→forAll(l | l.aula.nome = nome)	

Nome Metodo	+ aggiungiLezione(Lezione l) : void
Descrizione Metodo	Aggiunge o aggiorna una Lezione nel database.
Pre-condizioni	context LezioneDAO::aggiungiLezione(Lezione) : void
pre: l ≠ null	
Post-condizioni	context LezioneDAO::aggiungiLezione(Lezione) : void
post: LezioneDAO::trovaTutte()→includes(l)	

Nome Metodo	+ rimuoviLezione(Lezione l) : void
Descrizione Metodo	Rimuove una Lezione dal database.
Pre-condizioni	context LezioneDAO::rimuoviLezione(Lezione) : void
pre: l ≠ null and LezioneDAO::trovaTutte()→includes(l)	
Post-condizioni	context LezioneDAO::rimuoviLezione(Lezione) : void
post: not LezioneDAO::trovaTutte()→includes(l)	

Nome Metodo	+ trovaLezioniCorsoLaureaRestoAnno(long clid, long reid, int anid) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni relative a un corso di laurea, resto e anno.
Pre-condizioni	context Lezione-DAO::trovaLezioniCorsoLaureaRestoAnno(long, long, int) : List(Lezione)
pre: clid > 0 and reid > 0 and anid > 0	
Post-condizioni	context Lezione-DAO::trovaLezioniCorsoLaureaRestoAnno(long, long, int) : List(Lezione)
post: result→forAll(l | l.corsoLaurea.id = clid and l.resto.id = reid and l.anno.id = anid)	

Nome Metodo	+ trovaLezioniCorsoLaureaRestoAnnoSemestre(long clid, long reid, int anid, int semestre) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni relative a un corso di laurea, resto, anno e semestre.
Pre-condizioni	context Lezione-DAO::trovaLezioniCorsoLaureaRestoAnnoSemestre(long, long, int, int) : List(Lezione)
pre: clid > 0 and reid > 0 and anid > 0 and (semestre = 1 or semestre = 2)	
Post-condizioni	context Lezione-DAO::trovaLezioniCorsoLaureaRestoAnnoSemestre(long, long, int, int) : List(Lezione)
post: result→forAll(l | l.corsoLaurea.id = clid and l.resto.id = reid and l.anno.id = anid and l.semestre = semestre)	

Nome Metodo	+ trovaLezioniDocente(String nomeDocente) : List(Lezione)
Descrizione Metodo	Trova tutte le lezioni tenute da un determinato docente.
Pre-condizioni	context LezioneDAO::trovaLezioniDocente(String) : List(Lezione)
pre: nomeDocente ≠ null and nomeDocente.size() > 0	
Post-condizioni	context LezioneDAO::trovaLezioniDocente(String) : List(Lezione)
post: result→forAll(l | l.docenti→exists(d | d.nome = nomeDocente))	

Nome Interfaccia	RestoRemote
Scopo	Interfaccia relativa alla gestione dei vari resti all'interno dei corsi di laurea esistenti.
Invariante di classe	context Resto inv: self.nome ≠ null and self.nome.size() > 0

Nome Metodo	+ trovaRestiCorsoLaurea(CorsoLaurea corsoLaurea)
Descrizione Metodo	Trova i resti associati a un determinato corso di laurea.
Pre-condizioni	context RestoDAO::trovaRestiCorsoLaurea(corsoLaurea: CorsoLaurea) : List(Resto)
pre: corsoLaurea ≠ null and corsoLaurea.nome ≠ null and corsoLaurea.nome.size() > 0	
Post-condizioni	context RestoDAO::trovaRestiCorsoLaurea(corsoLaurea: CorsoLaurea) : List(Resto)
post: result→forAll(r | r.corsoLaurea = corsoLaurea)@pre.self.getUtenti().size() + 1	

Nome Metodo	+ trovaRestiCorsoLaurea(String nomeCorsoLaurea) : List<Resto>
Descrizione Metodo	Trova i resti associati a un determinato corso di laurea tramite il nome del corso.
Pre-condizioni	context RestoDAO::trovaRestiCorsoLaurea(nomeCorsoLaurea: String) : List(Resto)
pre: nomeCorsoLaurea ≠ null and nomeCorsoLaurea.size() > 0	
Post-condizioni	context RestoDAO::trovaRestiCorsoLaurea(nomeCorsoLaurea: String) : List(Resto)
post: result→forAll(r | r.corsoLaurea.nome = nomeCorsoLaurea)	

Nome Metodo	+ trovaResto(String nomeResto)
Descrizione Metodo	Trova un resto nel database tramite il nome del resto.
Pre-condizioni	context RestoDAO::trovaResto(nomeResto: String) : List(Resto)
pre: nomeResto ≠ null and nomeResto.size() > 0	
Post-condizioni	context RestoDAO::trovaResto(nomeResto: String) : List(Resto)
post: result→forAll(r | r.nome = nomeResto)	

Nome Metodo	+ trovaResto(long id)
Descrizione Metodo	Trova un resto nel database tramite l'ID.
Pre-condizioni	context RestoDAO::trovaResto(id: Long) : Resto
pre: id > 0	
Post-condizioni	context RestoDAO::trovaResto(id: Long) : Resto
post: result.id = id	

Nome Metodo	+ aggiungiResto(Resto resto)
Descrizione Metodo	Aggiunge o aggiorna un resto nel database.
Pre-condizioni	context RestoDAO::aggiungiResto(resto: Resto)
pre: resto ≠ null	
Post-condizioni	context RestoDAO::aggiungiResto(resto: Resto)
post: emUniClass.contains(resto)	

Nome Metodo	+ rimuoviResto(Resto resto)
Descrizione Metodo	Rimuove un resto dal database.
Pre-condizioni	context RestoDAO::rimuoviResto(resto: Resto)
pre: resto ≠ null and emUniClass.contains(resto)	
Post-condizioni	context RestoDAO::rimuoviResto(resto: Resto)
post: not emUniClass.contains(resto)	

3.2 Package Gestione Utenti

Nome Interfaccia	UserDirectory
Scopo	Interfaccia Facade per la gestione centralizzata degli utenti. Funge da punto di accesso unificato per i controller, disaccoppiandoli dalla complessità del modello dati e della logica di business.
Invariante di classe	Dipendenze: `UtenteService`

Nome Metodo	+ login(email: String, password: String) : Utente
Descrizione Metodo	Autentica un utente verificando le credenziali fornite.
Pre-condizioni	context UserDirectory::login(String, String) : Utente
pre: email ≠ null and password ≠ null	
Post-condizioni	context UserDirectory::login(String, String) : Utente
post: result ≠ null or thrown AuthenticationException	

Nome Metodo	+ getUser(email: String) : Utente
Descrizione Metodo	Recupera un utente tramite il suo indirizzo email. Restituisce null se non trovato.
Pre-condizioni	context UserDirectory::getUser(String) : Utente
pre: email ≠ null	
Post-condizioni	context UserDirectory::getUser(String) : Utente
post: result = null or result.email = email	

Nome Metodo	+ getAccademico(email: String) : Accademico
Descrizione Metodo	Recupera il profilo Accademico associato all'email. Restituisce null se l'utente non è di tipo Accademico.
Pre-condizioni	context UserDirectory::getAccademico(String) : Accademico
pre: email ≠ null	
Post-condizioni	context UserDirectory::getAccademico(String) : Accademico
post: result ≠ null implies result.oclIsTypeOf(Accademico)	

Nome Metodo	+ isDocente(email: String) : boolean
Descrizione Metodo	Verifica se l'utente identificato possiede il ruolo Docente.
Pre-condizioni	context UserDirectory::isDocente(String) : boolean
pre: email ≠ null	
Post-condizioni	context UserDirectory::isDocente(String) : boolean
post: result = (self.getAccademico(email).ruolo = Ruolo::DOCENTE)	

Nome Metodo	+ isStudente(email: String) : boolean
Descrizione Metodo	Verifica se l'utente identificato possiede il ruolo Studente.
Pre-condizioni	context UserDirectory::isStudente(String) : boolean
pre: email ≠ null	
Post-condizioni	context UserDirectory::isStudente(String) : boolean
post: result = (self.getAccademico(email).ruolo = Ruolo::STUDENTE)	

Nome Metodo	+ isCoordinatore(email: String) : boolean
Descrizione Metodo	Verifica se l'utente identificato possiede il ruolo Coordinatore.
Pre-condizioni	context UserDirectory::isCoordinatore(String) : boolean
pre: email ≠ null	
Post-condizioni	context UserDirectory::isCoordinatore(String) : boolean
post: result = (self.getAccademico(email).ruolo = Ruolo::COORDINATORE)	

Nome Metodo	+ getTuttiGliUtenti() : List(Utente)
Descrizione Metodo	Recupera la lista di tutti gli utenti registrati nel sistema.
Pre-condizioni	
Post-condizioni	context UserDirectory::getTuttiGliUtenti() : List(Utente)
post: result ≠ null	

Nome Metodo	+ getAccademiciPerRuolo(ruolo: Ruolo) : List(Accademico)
Descrizione Metodo	Recupera la lista degli accademici filtrati per un determinato ruolo.
Pre-condizioni	context UserDirectory::getAccademiciPerRuolo(Ruolo) : List(Accademico)
pre: ruolo ≠ null	
Post-condizioni	context UserDirectory::getAccademiciPerRuolo(Ruolo) : List(Accademico)
post: result→forAll(a | a.ruolo = ruolo)	

Nome Metodo	+ updateProfile(utente: Utente) : void
Descrizione Metodo	Aggiorna il profilo di un utente esistente.
Pre-condizioni	context UserDirectory::updateProfile(Utente) : void
pre: utente ≠ null	
Post-condizioni	context UserDirectory::updateProfile(Utente) : void
post: utenteService.aggiornaUtente(utente) executed	

Nome Metodo	+ cambiaStatoAttivazione(email: String, stato: boolean) : void
Descrizione Metodo	Modifica lo stato di attivazione di un account Accademico.
Pre-condizioni	context UserDirectory::cambiaStatoAttivazione(String, boolean) : void
pre: email ≠ null	
Post-condizioni	context UserDirectory::cambiaStatoAttivazione(String, boolean) : void
post: let u = self.getUser(email) in if u.oclIsTypeOf(Accademico) then u.attivato = stato endif	

Nome Interfaccia	UtenteService
Scopo	Servizio di business logic che gestisce le operazioni CRUD e di persistenza per le entità Utente e Accademico.
Invariante di classe	Dipendenze: `UtenteRemote`, `AccademicoRemote`

Nome Metodo	+ login(email: String, password: String) : Utente
Descrizione Metodo	Autentica un utente verificando le credenziali presso il DAO.
Pre-condizioni	context UtenteService::login(String, String) : Utente
pre: email ≠ null and password ≠ null	
Post-condizioni	context UtenteService::login(String, String) : Utente
post: result ≠ null or thrown AuthenticationException	

Nome Metodo	+ registraUtente(utente: Utente) : void
Descrizione Metodo	Registra un utente generico (es. Personale Amministrativo) nella tabella base Utente.
Pre-condizioni	context UtenteService::registraUtente(Utente) : void
pre: utente ≠ null and utenteDAO.findByEmail(utente.email) = null	
Post-condizioni	context UtenteService::registraUtente(Utente) : void
post: utenteDAO.contains(utente)	

Nome Metodo	+ registraAccademico(accademico: Accademico, ruolo: Ruolo) : void
Descrizione Metodo	Registra un nuovo accademico assegnandogli un ruolo specifico e persistendolo nella tabella estesa.
Pre-condizioni	context UtenteService::registraAccademico(Accademico, Ruolo) : void
pre: accademico ≠ null and ruolo ≠ null and utenteDAO.findByEmail(accademico.email) = null	
Post-condizioni	context UtenteService::registraAccademico(Accademico, Ruolo) : void
post: accademico.ruolo = ruolo and accademicoDAO.contains(accademico)	

Nome Metodo	+ getUtenteByEmail(email: String) : Utente
Descrizione Metodo	Recupera un utente generico tramite il suo indirizzo email.
Pre-condizioni	context UtenteService::getUtenteByEmail(String) : Utente
pre: email ≠ null	
Post-condizioni	context UtenteService::getUtenteByEmail(String) : Utente
post: result.email = email or thrown NotFoundUserException	

Nome Metodo	+ getAccademiciPerRuolo(ruolo: Ruolo) : List(Accademico)
Descrizione Metodo	Restituisce tutti gli accademici filtrati per ruolo.
Pre-condizioni	context UtenteService::getAccademiciPerRuolo(Ruolo) : List(Accademico)
pre: ruolo ≠ null	
Post-condizioni	context UtenteService::getAccademiciPerRuolo(Ruolo) : List(Accademico)
post: result→forAll(a | a.ruolo = ruolo)	

Nome Metodo	+ getTuttiGliUtenti() : List(Utente)
Descrizione Metodo	Recupera la lista di tutti gli utenti registrati nel sistema.
Pre-condizioni	
Post-condizioni	context UtenteService::getTuttiGliUtenti() : List(Utente)
post: result ≠ null	

Nome Metodo	+ aggiornaUtente(utente: Utente) : void
Descrizione Metodo	Aggiorna i dati di un utente. Gestisce il polimorfismo per scegliere il DAO corretto.
Pre-condizioni	context UtenteService::aggiornaUtente(Utente) : void
pre: utente ≠ null	
Post-condizioni	context UtenteService::aggiornaUtente(Utente) : void
post: if utente.oclIsTypeOf(Accademico) then accademicoDAO.updated(utente) else utenteDAO.updated(utente) endif	

Nota sulle rimozioni: Le interfacce `StudenteRemote`, `DocenteRemote`, `CoordinatoreRemote` e i relativi Service specifici sono stati rimossi in quanto ridondanti. La distinzione comportamentale è ora gestita tramite l'attributo `Ruolo` presente nella classe `Accademico`.

3.3 Package Gestione Conversazioni

Nome Interfaccia	TopicRemote
Scopo	Interfaccia relativa alla gestione dei topic di messaggistica.
Invariante di classe	

Nome Metodo	+ trovaId(id: long) : Topic
Descrizione Metodo	Trova un topic in base al suo ID.
Pre-condizioni	context topicDAO::trovaId(id: Long) : Topic
pre: id > 0	
Post-condizioni	context topicDAO::trovaId(id: Long) : Topic
post: result.id = id	

Nome Metodo	+ trovaNome(nome: String) : Topic
Descrizione Metodo	Trova un topic in base al suo nome.
Pre-condizioni	context topicDAO::trovaNome(nome: String) : Topic
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context topicDAO::trovaNome(nome: String) : Topic
post: result.nome = nome	

Nome Metodo	+ trovaCorsoLaurea(nome: String) : Topic
Descrizione Metodo	Trova un topic associato a un corso di laurea.
Pre-condizioni	context topicDAO::trovaCorsoLaurea(nome: String) : Topic
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context topicDAO::trovaCorsoLaurea(nome: String) : Topic
post: result.corsoLaurea.nome = nome	

Nome Metodo	+ trovaCorso(nome: String) : Topic
Descrizione Metodo	Trova un topic associato a un corso specifico.
Pre-condizioni	context topicDAO::trovaCorso(nome: String) : Topic
pre: nome ≠ null and nome.size() > 0	
Post-condizioni	context topicDAO::trovaCorso(nome: String) : Topic
post: result.corso.nome = nome	

Nome Metodo	+ trovaTutti() : List(Topic)
Descrizione Metodo	Restituisce una lista di tutti i topic disponibili.
Pre-condizioni	
Post-condizioni	context topicDAO::trovaTutti() : List(Topic)
post: result→forAll(t | t ≠ null)	

Nome Metodo	+ aggiungiTopic(topic: Topic) : void
Descrizione Metodo	Aggiunge un nuovo topic o aggiorna un topic esistente nel database.
Pre-condizioni	context topicDAO::aggiungiTopic(topic: Topic) : void
pre: topic ≠ null	
Post-condizioni	context topicDAO::aggiungiTopic(topic: Topic) : void
post: emUniClass.contains(topic)	

Nome Metodo	+ rimuoviTopic(topic: Topic) : void
Descrizione Metodo	Rimuove un topic dal database.
Pre-condizioni	context topicDAO::rimuoviTopic(topic: Topic) : void
pre: topic ≠ null and emUniClass.contains(topic)	
Post-condizioni	context topicDAO::rimuoviTopic(topic: Topic) : void
post: not emUniClass.contains(topic)	

Nome Interfaccia	MessaggioRemote
Scopo	Interfaccia relativa alla gestione dei messaggi.
Invariante di classe	context Messaggio inv: Messaggio.allInstances()→forAll(m | m.mittente ≠ null and m.destinatario ≠ null)

Nome Metodo	+ trovaMessaggio(id: long) : Messaggio
Descrizione Metodo	Trova un messaggio specifico dato il suo ID.
Pre-condizioni	context messaggioDAO::trovaMessaggio(id: Long) : Messaggio
pre: id > 0)	
Post-condizioni	context messaggioDAO::trovaMessaggio(id: Long) : Messaggio
post: result.id = id	

Nome Metodo	+ trovaMessaggiInviati(matricola: String) : List(Messaggio)
Descrizione Metodo	Recupera tutti i messaggi inviati da un determinato utente.
Pre-condizioni	context messaggioDAO::trovaMessaggiInviati(matricola: String) : List(Messaggio)
pre: matricola ≠ null and matricola.size() > 0	
Post-condizioni	context messaggioDAO::trovaMessaggiInviati(matricola: String) : List(Messaggio)
post: result→forAll(m | m.mittente = matricola)	

Nome Metodo	+ trovaMessaggiRicevuti(matricola: String) : List(Messaggio)
Descrizione Metodo	Recupera tutti i messaggi ricevuti da un determinato utente.
Pre-condizioni	context messaggioDAO::trovaMessaggiRicevuti(matricola: String) : List(Messaggio)
pre: matricola ≠ null and matricola.size() > 0	
Post-condizioni	context messaggioDAO::trovaMessaggiRicevuti(matricola: String) : List(Messaggio)
post: result→forAll(m | m.destinatario = matricola)	

Nome Metodo	+ trovaMessaggi(matricola1: String, matricola2: String) : List(Messaggio)
Descrizione Metodo	Recupera i messaggi scambiati tra due utenti.
Pre-condizioni	context messaggioDAO::trovaMessaggi(matricola1: String, matricola2: String) : List(Messaggio)
pre: matricola1 ≠ null and matricola2 ≠ null and matricola1 ≠ matricola2	
Post-condizioni	context messaggioDAO::trovaMessaggi(matricola1: String, matricola2: String) : List(Messaggio)
post: result→forAll(m | (m.mittente = matricola1 and m.destinatario = matricola2) or (m.mittente = matricola2 and m.destinatario = matricola1))	

Nome Metodo	+ trovaTutti() : List(Messaggio)
Descrizione Metodo	Recupera tutti i messaggi presenti nel database.
Pre-condizioni	
Post-condizioni	context messaggioDAO::trovaTutti() : List(Messaggio)
post: result→forAll(m | m ≠ null)	

Nome Metodo	+ trovaAvvisi() : List(Messaggio)
Descrizione Metodo	Recupera tutti gli avvisi presenti nel sistema.
Pre-condizioni	
Post-condizioni	context messaggioDAO::trovaAvvisi() : List(Messaggio)
post: result→forAll(m | m.tipo = 'Avviso')	

Nome Metodo	+ trovaAvvisiAutore(autore: String) : List(Messaggio)
Descrizione Metodo	Recupera tutti gli avvisi presenti nel sistema con un certo autore.
Pre-condizioni	context messaggioDAO::trovaAvvisiAutore(autore: String) : List(Messaggio)
pre: autore $\neq$ null and autore.size() $>$ 0	
Post-condizioni	context messaggioDAO::trovaAvvisiAutore(autore: String) : List(Messaggio)
post: result→forAll(m | m.autore = autore and m.tipo = 'Avviso')	

Nome Metodo	+ trovaMessaggiData(dateTime: LocalDateTime) : List(Messaggio)
Descrizione Metodo	Recupera tutti i messaggi inviati in una determinata data.
Pre-condizioni	context messaggioDAO::trovaMessaggiData(dateTime: LocalDateTime) : List(Messaggio)
pre: dateTime $\neq$ null	
Post-condizioni	context messaggioDAO::trovaMessaggiData(dateTime: LocalDateTime) : List(Messaggio)
post: result→forAll(m | m.dataInvio = dateTime)	

Nome Metodo	+ trovaTopic(topic: Topic) : List(Messaggio)
Descrizione Metodo	Recupera tutti i messaggi appartenenti a un determinato topic.
Pre-condizioni	context messaggioDAO::trovaTopic(topic: Topic) : List(Messaggio)
pre: topic $\neq$ null	
Post-condizioni	context messaggioDAO::trovaTopic(topic: Topic) : List(Messaggio)
post: result→forAll(m | m.topic = topic)	

Nome Metodo	+ aggiungiMessaggio(messaggio: Messaggio) : Messaggio
Descrizione Metodo	Aggiunge un nuovo messaggio o aggiorna un messaggio esistente.
Pre-condizioni	context messaggioDAO::aggiungiMessaggio(messaggio: Messaggio) : Messaggio
pre: messaggio $\neq$ null	
Post-condizioni	context messaggioDAO::aggiungiMessaggio(messaggio: Messaggio) : Messaggio
post: emUniClass.contains(messaggio)	

Nome Metodo	+ rimuoviMessaggio(messaggio: Messaggio) : void
Descrizione Metodo	Rimuove un messaggio dal database.
Pre-condizioni	context messaggioDAO::rimuoviMessaggio(messaggio: Messaggio) : void
pre: messaggio $\neq$ null and emUniClass.contains(messaggio)	
Post-condizioni	context messaggioDAO::rimuoviMessaggio(messaggio: Messaggio) : void
post: not emUniClass.contains(messaggio)	