Università degli Studi di Salerno

Corso di Ingegneria del Software Tecniche Avanzate

UniClass

Refactoring Architetturale e Impact Analysis

Gruppo di Lavoro:

Cognome e Nome	**Matricola**
Cammarota Lucageneroso	NF22500053
Sabetta Giuseppe	NF22500155

12 febbraio 2026

Registro delle Modifiche

Vers.	Data	Autore	Descrizione Modifica
1.0	06/02/2026	Gruppo	Stesura iniziale, definizione Candidate Impact Set (CIS).
1.0	07/02/2026	Gruppo	Analisi dei grafi delle dipendenze e refactoring DAO.
1.0	08/02/2026	Gruppo	Definizione strategia di migrazione dati e scheduling Bottom-Up.
1.0	08/02/2026	Gruppo	Approvazione finale per l'implementazione della CR.

Impact Analysis Report

Refactoring del Modulo Utenti e Razionalizzazione del Modello Dati

UniClass – Architettura e Manutenzione Evolutiva

12 febbraio 2026

Indice

1 Introduzione

Il presente documento analizza in modo sistematico l'impatto del refactoring del modulo *Utenti* del sistema UniClass. L'intervento si colloca all'interno di un più ampio processo di razionalizzazione architetturale, volto a superare le limitazioni del modello basato su ereditarietà e a introdurre una struttura fondata sulla composizione. Tale revisione consente di distinguere in modo più netto le responsabilità delle entità *Utente*, *Accademico* e *Personale Tecnico-Amministrativo*, migliorando la coerenza semantica del dominio e riducendo l'accoppiamento tra i moduli.

L'analisi si concentra sull'identificazione del *Candidate Impact Set* (CIS), sulla valutazione degli impatti architetturali e sul disegno di una strategia di regression testing adeguata a garantire la continuità operativa del sistema.

Al termine della sezione, si riportano i punti essenziali:

- il refactoring modifica il modello dati e la struttura dei servizi;

- l'obiettivo è ridurre l'accoppiamento e migliorare la manutenibilità;
- il CIS costituisce la base per la valutazione dell'impatto.

2 Descrizione del Cambiamento

Il refactoring interviene sul modello dati e sulla logica applicativa del modulo Utenti. La precedente gerarchia basata su ereditarietà presentava ridondanze, campi nulli e una proliferazione di servizi specializzati, con conseguente aumento della complessità. La nuova impostazione introduce un modello più lineare, in cui l'entità *Utente* assume il ruolo di contenitore principale, mentre l'entità *Accademico* rappresenta un'estensione opzionale, collegata tramite una relazione uno-a-uno.

La revisione comporta anche la rimozione dei servizi specifici (StudenteService, DocenteService, CoordinatoreService), sostituiti da un accesso centralizzato tramite *UserDirectory*. Tale scelta riduce la dispersione delle responsabilità e semplifica l'interazione tra moduli.

In sintesi:

- il modello dati viene semplificato tramite composizione;
- i servizi specializzati vengono eliminati o accorpati;
- l'accesso ai dati utente viene centralizzato.

2.1 Change Request e Mapping sui Deliverable Esistenti

La presente impact analysis è guidata da una *Change Request* finalizzata al refactoring del modulo Utenti, con l'obiettivo di superare le limitazioni emerse nei deliverable originali dell'applicazione. In particolare, la richiesta di cambiamento non introduce nuovi requisiti funzionali, ma interviene sulla struttura architetturale e sul modello dati al fine di migliorare manutenibilità, coerenza semantica e controllo dell'impatto evolutivo.

La Tabella 2 mette in relazione gli elementi principali della Change Request con i deliverable legacy, evidenziandone lo stato e il tipo di impatto.

Change Request Item	**Deliverable Legacy**	**Stato**	**Tipo di Impatto**
Eliminazione gerarchia Utente	Modello ER v1	Modificato	Strutturale
Accorpamento servizi specializzati	Design Architetturale v1	Rimosso	Architetturale

Change Request Item	Deliverable Legacy	Stato	Tipo di Impatto
Centralizzazione accesso utenti	Specifica Servizi v1	Sostituito	Contrattuale
Flussi di login	UC1 – Login	Invariato	Nessuno
Gestione sicurezza accessi	NFR1 – Sicurezza	Rafforzato	Non funzionale

Tabella 2: Mapping tra Change Request e deliverable legacy

L'analisi dimostra come il cambiamento si configuri come un'evoluzione controllata del progetto originale, preservando i requisiti funzionali esistenti e intervenendo esclusivamente sui meccanismi di implementazione e integrazione.

2.2 Starting Impact Set (SIS)

In accordo con il processo di Impact Analysis descritto in [**?**], lo *Starting Impact Set (SIS)* per la Change Request *CR-UniClass* è stato identificato a partire dagli artefatti esplicitamente coinvolti nel *Reason for Change* e nella *Description of Change*.

In particolare, lo SIS comprende:

- **Requisiti (RAD)**: UC1 – Autenticazione, UC9 – Creazione Account UniClass, NFR1 – Sicurezza.
- **Design (SDD/ODD)**: Object Model delle gerarchie di utenza (pacchetto `it.unisa.uniclass.ut` Sequence Diagram SD1, SD7, SD9, SD10 relativi a login, creazione account e gestione utenza.
- **Codice (Implementation)** suddiviso per layer architetturale:
  - **Control (Servlet Layer)**:
    - `LoginServlet`
    - `GetEmailServlet`
    - `AttivaUtentiServlet`
  - **Service Layer**:
    - `AccademicoService`
    - `UtenteService`
    - `DocenteService`

    * `StudenteService`
    * `CoordinatoreService`
  - **Model Layer (Gerarchia da collassare)**:
    * `Utente`
    * `Accademico`
    * `Studente`
    * `Docente`
    * `Coordinatore`
    * `PersonaleTA`
  - **DAO Layer**:
    * `AccademicoDAO`
    * `UtenteDAO`
    * `DocenteDAO`
    * `StudenteDAO`
    * `CoordinatoreDAO`

- **Test**: `AccademicoServiceTest`, `LoginServletTest`, test di regressione associati ai casi d'uso UC1 e UC9.

Su questo SIS è stata poi condotta l'analisi delle dipendenze per derivare il *Candidate Impact Set (CIS)*, distinguendo tra dipendenze backward e forward.

3 Metodologia di Analisi e Costruzione del CIS

La definizione del *Candidate Impact Set* è stata condotta attraverso un processo di reverse engineering, basato sull'analisi delle backward dependencies delle componenti coinvolte nello SIS. Tale approccio consente di individuare tutti i componenti che dipendono direttamente o indirettamente dalle classi che sono direttamente oggetto di refactoring.

L'analisi ha riguardato:

- i servizi del modulo Utenti destinati alla rimozione o modifica;
- i controller che invocano tali servizi;
- le JSP che ne utilizzano i dati;
- i test unitari, di integrazione e di benchmark;
- le componenti DAO e Remote correlate.

I grafi completi delle backward dependencies sono riportati in Appendice A.

4 Candidate Impact Set (CIS)

Il CIS risultante dall'analisi combinata delle backward e forward dependencies dei servizi del modulo *Utenti* è riportato nella Tabella **??**. La tabella integra le dipendenze già individuate nella fase di reverse engineering con le nuove dipendenze forward emerse dall'analisi statica del codice.

Componente	**Package**	**Backward Dependencies**
AccademicoService	controller	LoginServlet, AttivaUtentiServlet, GetAttivati, GetNonAttivati, GetEmailServlet.
AccademicoService	controller (conversazioni)	chatServlet, invioMessaggioServlet, ConversazioniServlet.
AccademicoService	test	AccademicoServiceTest, StudenteServiceTest, DocenteServiceTest, CoordinatoreServiceTest, UtenteServiceTest, vari test di controller.
AccademicoService	service	Dipendenze indirette da StudenteService, DocenteService, CoordinatoreService, UtenteService.
AccademicoService	model	–
AccademicoService	dao / remote	–
AccademicoService	exceptions	–
AccademicoService	jdk / ejb	–
StudenteService	jsp	Account.jsp.
StudenteService	test	StudenteServiceTest.
StudenteService	service	–
StudenteService	model	–
StudenteService	dao / remote	–
StudenteService	exceptions	–
StudenteService	jdk	–
DocenteService	jsp	Account.jsp.
DocenteService	test	DocenteServiceTest, DocenteServiceBenchmark.
DocenteService	service	–
DocenteService	model	–
DocenteService	dao / remote	–
DocenteService	exceptions	–

Componente	Package	Backward Dependencies
DocenteService	jdk	–
CoordinatoreService	jsp	Account.jsp.
CoordinatoreService	test	CoordinatoreServiceTest, CoordinatoreServiceBenchmark.
CoordinatoreService	service	–
CoordinatoreService	model	–
CoordinatoreService	dao / remote	–
CoordinatoreService	exceptions	–
CoordinatoreService	jdk	–
PersonaleTAService	controller	LoginServlet.
PersonaleTAService	jsp	Account.jsp.
PersonaleTAService	service	UtenteService; test unitari e benchmark.
PersonaleTAService	model	–
PersonaleTAService	dao / remote	–
PersonaleTAService	exceptions	–
PersonaleTAService	jdk	–
UtenteService	test	UtenteServiceTest; benchmark JMH.
UtenteService	service	–
UtenteService	model	–
UtenteService	dao	–
UtenteService	exceptions	–
UtenteService	jdk / ejb	–

Tabella 3: Backward dependencies per componente e package

Componente	Package	Forward Dependencies
AccademicoService	controller	–
AccademicoService	controller (conversazioni)	–
AccademicoService	test	–
AccademicoService	service	–
AccademicoService	model	Accademico, Utente.
AccademicoService	dao / remote	AccademicoRemote.

Componente	Package	Forward Dependencies
AccademicoService	exceptions	NoResultException (JPA), RuntimeException.
AccademicoService	jdk / ejb	InitialContext, NamingException, List, String, Object, PrintStream, Stateless.
StudenteService	jsp	–
StudenteService	test	–
StudenteService	service	AccademicoService.
StudenteService	model	Studente, Accademico, Utente, CorsoLaurea.
StudenteService	dao / remote	StudenteRemote.
StudenteService	exceptions	AlreadyExistentUserException, NotFoundUserException, IncorrectUserSpecification, NoResultException (JPA), RuntimeException.
StudenteService	jdk	InitialContext, NamingException, List, String, Object.
DocenteService	jsp	–
DocenteService	test	–
DocenteService	service	AccademicoService.
DocenteService	model	Docente, Accademico, Utente.
DocenteService	dao / remote	DocenteRemote.
DocenteService	exceptions	AlreadyExistentUserException, NotFoundUserException, IncorrectUserSpecification, NoResultException (JPA), RuntimeException.
DocenteService	jdk	InitialContext, NamingException, List, String, Object.
CoordinatoreService	jsp	–
CoordinatoreService	test	–
CoordinatoreService	service	AccademicoService.
CoordinatoreService	model	Coordinatore, Accademico, Utente.
CoordinatoreService	dao / remote	CoordinatoreRemote.
CoordinatoreService	exceptions	AlreadyExistentUserException, NotFoundUserException, IncorrectUserSpecification, NoResultException (JPA), RuntimeException.

Componente	Package	Forward Dependencies
CoordinatoreService	jdk	InitialContext, NamingException, List, String, Object.
PersonaleTAService	controller	–
PersonaleTAService	jsp	–
PersonaleTAService	service	–
PersonaleTAService	model	PersonaleTA, Accademico.
PersonaleTAService	dao / remote	PersonaleTARemote.
PersonaleTAService	exceptions	NoResultException (JPA), RuntimeException, Exception.
PersonaleTAService	jdk	InitialContext, NamingException, List, String, Object, Stateless.
UtenteService	test	–
UtenteService	service	AccademicoService, PersonaleTAService.
UtenteService	model	PersonaleTA, Accademico, Utente.
UtenteService	dao	AccademicoDAO.
UtenteService	exceptions	AuthenticationException, NoResultException (JPA).
UtenteService	jdk / ejb	String, EJB, Stateless.

Tabella 4: Forward dependencies per componente e package

4.1 Classificazione dell'Impatto

La classificazione del Candidate Impact Set in *Primary* e *Secondary Impact* deriva direttamente dall'analisi dei grafi di dipendenza costruiti nella fase di reverse engineering, ed è fondata sullo studio delle **backward dependencies** (componenti chiamanti) e delle **forward dependencies** (componenti chiamati) dei servizi del modulo Utenti.

Per i componenti software, in particolare per i file di codice, la nozione di impatto è quindi definita in termini di *relazioni di chiamata* e non su valutazioni puramente qualitative o architetturali.

4.1.1 Modello di Dipendenza

Sia $G = (V, E)$ il grafo delle dipendenze, dove:

- ogni nodo $v \in V$ rappresenta un componente software (classe, servizio, controller, DAO);

- ogni arco diretto $(v_i, v_j) \in E$ indica una relazione di chiamata da v_i a v_j.

Dato un componente c, si definiscono:

- **Backward Dependencies** ($BD(c)$): l'insieme dei nodi che invocano c (chiamanti);
- **Forward Dependencies** ($FD(c)$): l'insieme dei nodi invocati da c (chiamati).

Il Candidate Impact Set è ottenuto come unione dei nodi raggiungibili tramite $BD(c)$ e $FD(c)$ a partire dai componenti direttamente coinvolti nella Change Request.

4.1.2 Criteri di Identificazione del Primary Impact Set

Un componente è incluso nel **Primary Impact Set** se soddisfa almeno uno dei seguenti criteri, derivati dall'analisi delle dipendenze:

P1. Origine del cambiamento: il componente è direttamente oggetto di modifica o rimozione secondo la Change Request.

P2. Elevato grado di dipendenza: il componente presenta un numero significativo di backward dependencies, risultando un nodo ad alta fan-in.

P3. Ruolo di snodo: il componente collega insiemi distinti di chiamanti e chiamati, fungendo da punto di propagazione dell'impatto.

P4. Ridefinizione delle chiamate: il componente modifica l'insieme delle forward dependencies a seguito del refactoring.

Tali componenti costituiscono il nucleo del refactoring e rappresentano i principali generatori del ripple effect osservato nei grafi di dipendenza.

Componente	**Motivazione (dipendenze)**	**Criteri**
AccademicoService	Nodo ad alto fan-in: numerosi controller e servizi lo invocano; variazione delle forward dependencies dovuta all'accorpamento dei servizi legacy.	P1, P2, P4
UserDirectory	Nuovo punto di chiamata introdotto; modifica delle catene di invocazione verso il modello Utenti.	P1, P3, P4
StudenteService, DocenteService, CoordinatoreService	Componenti rimossi: le loro backward dependencies vengono riallocate verso AccademicoService.	P1, P2

Componente	Motivazione (dipendenze)	Criteri
Modello dati Utente/Accademico	Ridefinizione delle relazioni che influenza le chiamate DAO e Service.	P1, P4

Tabella 5: Primary Impact Set basato su forward e backward dependencies

4.1.3 Criteri di Identificazione del Secondary Impact Set

Un componente è incluso nel **Secondary Impact Set** se:

S1. appartiene all'insieme delle backward dependencies di un componente primario (chiamante);

S2. utilizza risultati o dati prodotti da un componente primario (chiamato);

S3. non modifica la propria struttura interna, ma può subire variazioni comportamentali per effetto delle nuove catene di chiamata;

S4. richiede validazione tramite regression testing.

Il Secondary Impact Set rappresenta l'insieme dei componenti coinvolti per propagazione dell'impatto, secondo le relazioni di chiamata emerse dai grafi.

Componente	Relazione di dipendenza	Criteri
LoginServlet	Backward dependency di AccademicoService: invoca metodi di autenticazione e recupero dati.	S1, S4
AttivaUtentiServlet	Backward dependency di AccademicoService: utilizza i nuovi flussi di attivazione.	S1, S3
GetEmailServlet	Backward dependency; dipende dai dati forniti da AccademicoService.	S1, S2
JSP Account	Dipendenza indiretta: utilizza dati restituiti dai controller.	S2, S3
Test di integrazione e benchmark	Validano le catene di chiamata modificate.	S1, S4

Tabella 6: Secondary Impact Set basato su relazioni di chiamata

4.1.4 Sintesi del Processo di Costruzione

Il processo di costruzione del CIS e della sua classificazione è quindi riassumibile come segue:

- identificazione dei componenti direttamente coinvolti nella Change Request;
- costruzione dei grafi di forward e backward dependencies;
- individuazione dei nodi a maggiore fan-in e fan-out;
- classificazione dei nodi centrali come Primary Impact Set;
- inclusione dei nodi adiacenti nel Secondary Impact Set;
- validazione tramite tracciabilità verticale e orizzontale.

Questa impostazione garantisce che la classificazione dell'impatto sia oggettiva, riproducibile e direttamente derivata dall'analisi delle dipendenze di chiamata.

5 CIS per Package – Focus su AccademicoService

La seguente tabella approfondisce le dipendenze verso `AccademicoService`, suddividendole per package.

Componente	**Package**	**Dipendenze verso AccademicoService**
LoginServlet	controller	Utilizza AccademicoService per autenticazione e recupero informazioni utente.
AttivaUtentiServlet	controller	Invoca AccademicoService per attivazione utenti accademici.
GetAttivati	controller	Recupera elenco utenti accademici attivati.
GetNonAttivati	controller	Recupera elenco utenti accademici non attivati.
GetEmailServlet	controller	Recupera informazioni accademiche associate all'utente.
ConversazioniServlet	controller (conversazioni)	Valida mittente e destinatari tramite AccademicoService.
StudenteService	service	Dipendenza diretta: delega parte della logica ad AccademicoService.

Componente	**Package**	**Dipendenze verso AccademicoService**
DocenteService	service	Dipendenza diretta: eredita logiche accademiche.
CoordinatoreService	service	Dipendenza diretta: utilizza AccademicoService per la gestione del ruolo.
UtenteService	service	Dipendenza indiretta: richiama AccademicoService per operazioni accademiche.
AccademicoDAO	dao	Dipendenza strutturale per operazioni CRUD.
AccademicoRemote	remote	Interfaccia remota utilizzata da AccademicoService.
AccademicoServiceTest	test	Test diretto delle funzionalità.
Vari test controller	test	Dipendenze indirette tramite flussi applicativi.
Conversazioni.jsp, chat.jsp	jsp	Richiedono dati ottenuti tramite AccademicoService.

Tabella 7: CIS suddiviso per package per le dipendenze verso `AccademicoService`

6 Analisi dell'Impatto Architetturale

Il refactoring del modulo Utenti produce un impatto significativo sulla struttura architetturale del sistema, poiché modifica sia il modello dati sia il modo in cui i moduli applicativi interagiscono con le informazioni relative agli utenti. La transizione da una gerarchia basata su ereditarietà a un modello fondato sulla composizione consente di ottenere una rappresentazione più fedele del dominio, riducendo al contempo la complessità interna e migliorando la separazione delle responsabilità.

La Figura 1 illustra il nuovo modello architetturale, evidenziando la centralità della facciata *UserDirectory* come punto di accesso unificato ai dati utente. Tale componente svolge un ruolo determinante nel contenere l'impatto del refactoring, poiché isola i moduli esterni dalla struttura interna del dominio Utenti. I servizi applicativi, come *ConversazioniService* e *OrarioService*, non interagiscono più direttamente con i servizi specializzati (StudenteService, DocenteService, CoordinatoreService), ma si affidano a un'interfaccia stabile e coerente.

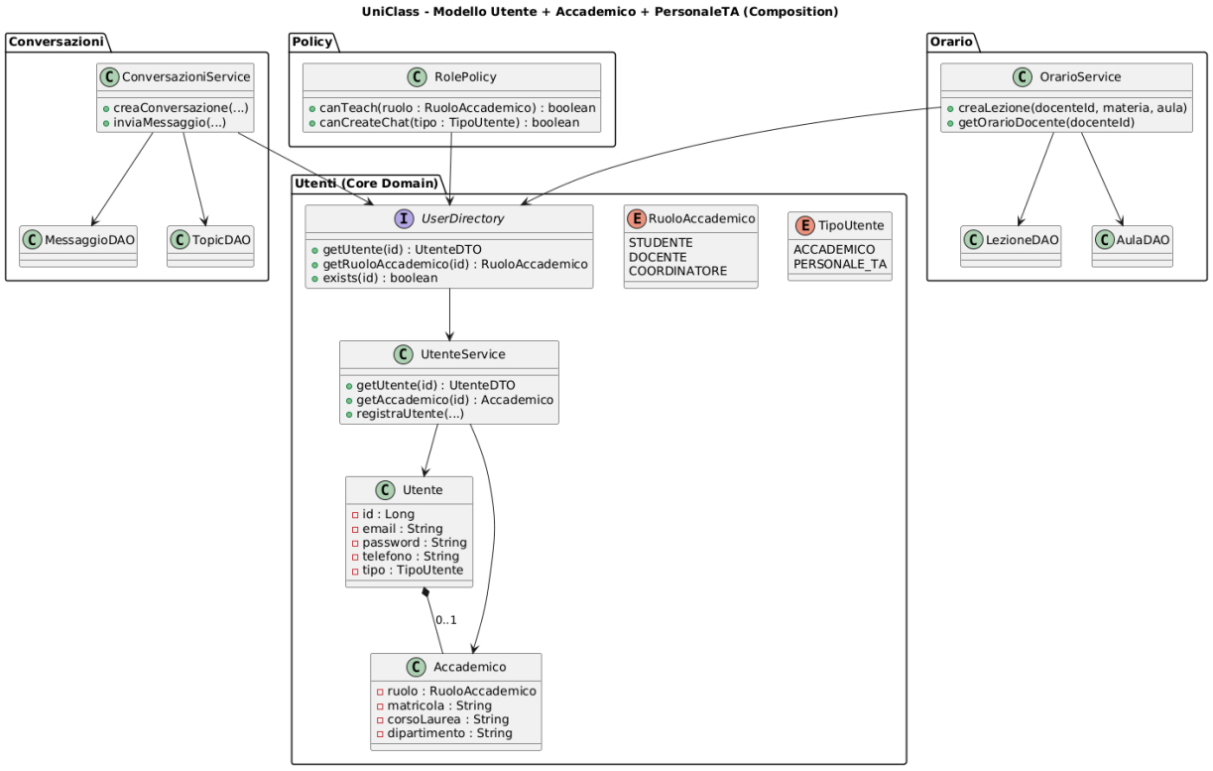


Figura 1: Modello architetturale aggiornato basato su composizione per la gestione degli utenti

Il diagramma mette in evidenza tre aspetti fondamentali. In primo luogo, la semplificazione del dominio Utenti, ottenuta attraverso la distinzione tra l'entità *Utente* e la sua estensione opzionale *Accademico*, consente di eliminare la proliferazione di servizi specializzati e di ridurre la ridondanza logica. In secondo luogo, la presenza di un'interfaccia unificata permette di disaccoppiare i moduli applicativi dal modello dati, garantendo una maggiore stabilità rispetto alle modifiche interne. Infine, la riorganizzazione dei flussi di accesso ai dati produce un impatto positivo sulla manutenibilità complessiva del sistema, poiché riduce il numero di punti di integrazione e semplifica la gestione delle dipendenze.

In sintesi:

- il modello architetturale risulta più coerente e meno soggetto a ripple effect;
- la facciata *UserDirectory* diventa il principale punto di isolamento tra moduli;
- la composizione sostituisce l'ereditarietà, riducendo complessità e ridondanza.

7 Impatto sul modello dati

L'analisi dell'impatto non può prescindere da una rappresentazione strutturale del modello dati e del modello di retrieval, poiché entrambi costituiscono il fondamento architetturale su cui si innesta il refactoring del modulo Utenti. La comprensione delle relazioni tra

le entità persistenti e dei flussi di accesso ai dati consente di valutare con precisione la portata delle modifiche introdotte e di identificare i punti del sistema maggiormente esposti a regressioni.

La Figura 2 presenta il modello Entity–Relationship aggiornato, che evidenzia la distinzione tra l'entità *UTENTE* e la sua estensione opzionale *ACCADEMICO*. Tale rappresentazione mette in luce la semplificazione ottenuta attraverso la rimozione della gerarchia ereditaria e la sua sostituzione con una relazione uno-a-uno, più coerente con il dominio applicativo e più robusta rispetto alle esigenze di evoluzione futura.

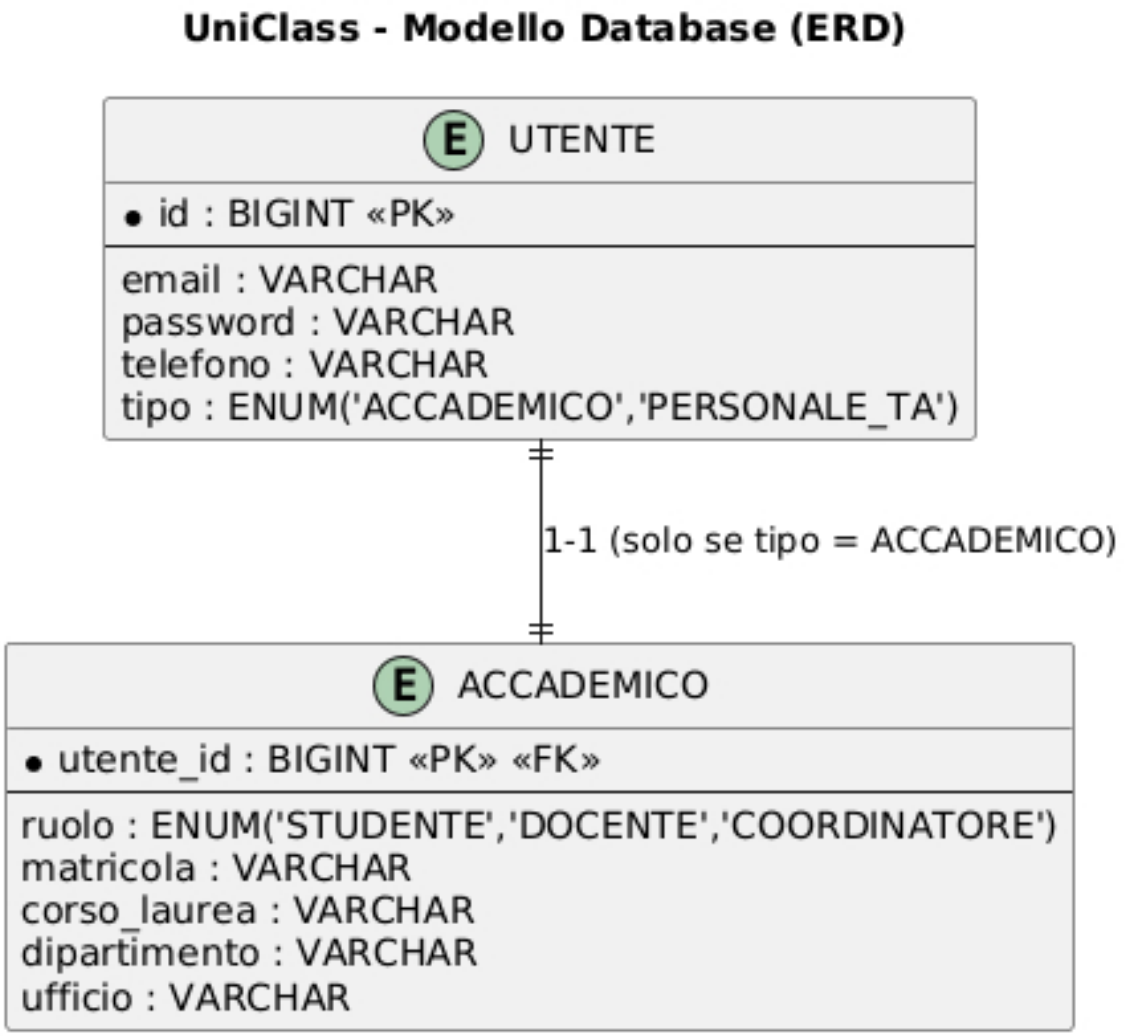


Figura 2: Modello ER aggiornato del dominio Utenti

La Figura 3 illustra invece il modello di retrieval, che descrive il percorso dei dati dal livello di persistenza fino ai servizi applicativi. La presenza di una facciata unificata, *UserDirectory*, consente di isolare i moduli esterni dalla complessità interna del dominio Utenti, riducendo l'accoppiamento e semplificando la gestione delle dipendenze. Tale struttura assume un ruolo centrale nell'analisi dell'impatto, poiché rappresenta il principale punto di interazione tra il refactoring e i moduli Conversazioni, Orario e Policy.

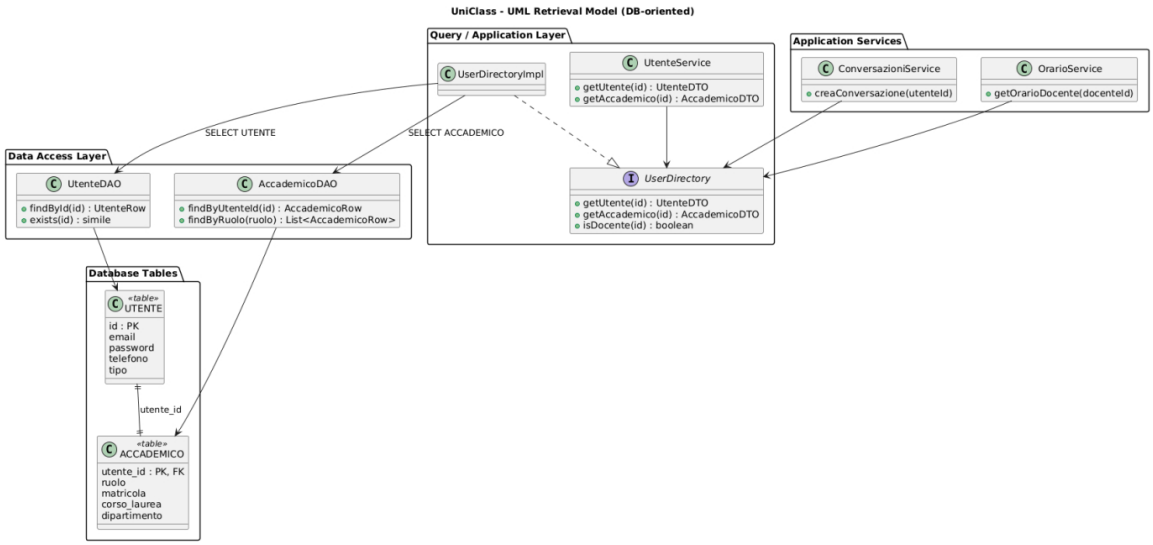


Figura 3: Modello di retrieval dei dati utente

Nel loro insieme, i due modelli forniscono una visione integrata del dominio e dei flussi informativi, permettendo di valutare con precisione le conseguenze del refactoring. L'impatto principale riguarda la sostituzione dei servizi specializzati con un accesso centralizzato, la riorganizzazione delle entità persistenti e la conseguente modifica dei punti di integrazione con i moduli esterni. La presenza di un'interfaccia unificata riduce la propagazione delle modifiche, ma richiede un'attenta verifica dei flussi di retrieval per garantire la continuità funzionale.

In sintesi, l'impact model evidenzia tre aspetti fondamentali:

- la coerenza strutturale introdotta dal nuovo modello dati;
- la centralità della facciata *UserDirectory* come punto di isolamento architetturale;
- la necessità di verificare i flussi di retrieval per assicurare la compatibilità con i moduli dipendenti.

8 Strategia di Migrazione Dati e Preservazione dell'Integrità

Il refactoring del modello dati, che prevede il passaggio da una gerarchia profonda (modello *Is-A* rigido) a una struttura composita basata su ruoli (modello ibrido `Utente-Accademico`), comporta una modifica sostanziale dello schema relazionale sottostante. Per garantire la continuità operativa del sistema *UniClass* e prevenire la perdita di informazioni storiche, è stata definita una procedura di migrazione dati strutturata secondo il paradigma ETL (Extract, Transform, Load).

8.1 Analisi delle Trasformazioni dello Schema

L'intervento richiede la fusione delle tabelle specializzate (`studente`, `docente`, `personale_ta`) nelle due nuove entità target. Le trasformazioni necessarie sono mappate come segue:

- **Consolidamento degli Attributi Comuni:** Attributi precedentemente frammentati nelle sottoclassi, quali `telefono`, `dipartimento` e `ufficio`, verranno promossi alla tabella principale `utente`. Ciò elimina la necessità di join multiple per il recupero di informazioni anagrafiche di base, riducendo il costo computazionale delle query di login.
- **Unificazione dei Ruoli Accademici:** Le entità `Studente`, `Docente` e `Coordinatore` convergeranno nella singola tabella `accademico`. La distinzione semantica sarà demandata a una nuova colonna discriminante (Enum `ruolo`), permettendo a un singolo record fisico di mutare logico (es. da Studente a Docente) senza operazioni di *DELETE/INSERT*.

8.2 Procedura di Migrazione

La migrazione avverrà mediante script SQL transazionali per assicurare l'atomicità dell'operazione. Si evidenzia la criticità dell'aggiornamento del componente `DatabasePopulator.java`, responsabile dell'inizializzazione dell'ambiente di test e sviluppo.

1. **Schema Evolution:** Modifica delle DDL per aggiungere la colonna discriminatore (`tipo`) sulla tabella `utente` e la colonna `ruolo` sulla tabella `accademico`.
2. **Data Transformation:**
   - Migrazione dei record da `personale_ta` a `utente`, valorizzando il discriminatore a 'PTA'.
   - Migrazione e unione dei record da `studente` e `docente` verso la tabella `accademico`, mappando le matricole e i codici identificativi sulla nuova colonna generica `matricola`.
3. **Constraint Update:** Aggiornamento delle chiavi esterne per puntare alle nuove tabelle consolidate, garantendo che le relazioni con entità esterne (es. `Corso`, `Lezione`) rimangano consistenti.

9 Horizontal Traceability

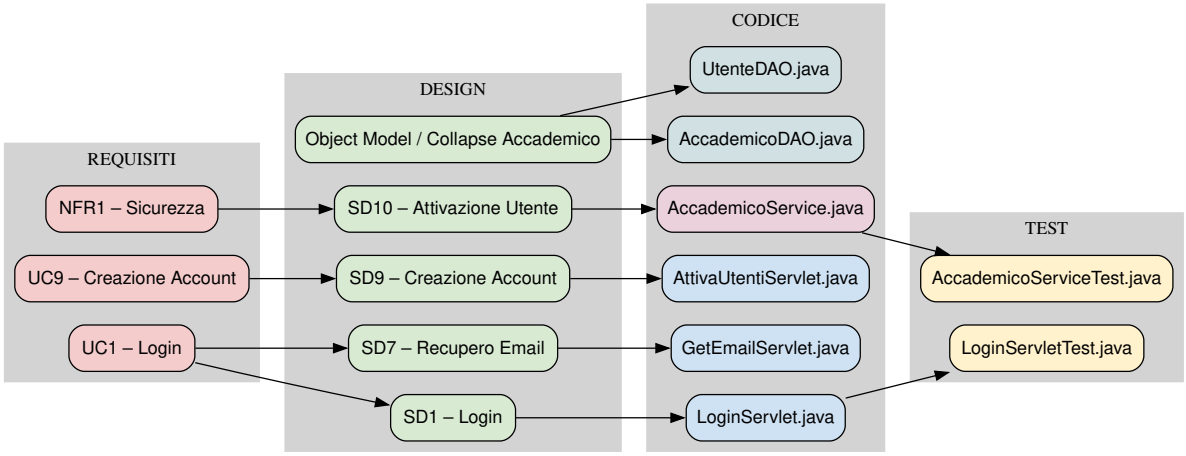


Figura 4: Horizontal Traceability Graph – UniClass Impact Analysis

9.1 Interpretazione dell'Horizontal Traceability Graph

L' *Horizontal Traceability Graph* rappresenta la tracciabilità multilivello che collega i requisiti impattati dalla Change Request (RAD), gli artefatti di design (SDD/ODD), le componenti software coinvolte (Implementation) e i relativi test di verifica (TP). Tale rappresentazione consente di evidenziare la propagazione del cambiamento lungo l'intero ciclo di vita del software, mostrando come una modifica concettuale alla gerarchia dell'utenza si rifletta progressivamente sui livelli inferiori.

Nel livello dei **Requisiti**, i casi d'uso UC1 (Autenticazione) e UC9 (Creazione Account), insieme al requisito non funzionale NFR1 (Sicurezza), costituiscono il punto di ingresso del cambiamento. Essi sono direttamente impattati dalla rimozione della gerarchia rigida `Accademico` e dalla necessità di introdurre un modello a ruoli dinamici.

Il livello di **Design** mostra gli artefatti che descrivono il comportamento e la struttura del sistema: l'Object Model della gerarchia utenti e i Sequence Diagram SD1, SD7, SD9 e SD10. Questi diagrammi modellano i flussi di login, recupero email, creazione e attivazione dell'account, tutti dipendenti dalla struttura dell'utenza e quindi direttamente coinvolti nel redesign.

Il livello di **Implementazione** evidenzia le classi che realizzano tali funzionalità: i controller (`LoginServlet`, `GetEmailServlet`, `AttivaUtentiServlet`), i service (`AccademicoService`, `UtenteService`, `DocenteService`, `StudenteService`, `CoordinatoreService`), i DAO e le entità del modello dati. La presenza di tali componenti nel grafo deriva dal fatto che essi implementano o dipendono direttamente dai flussi descritti nei Sequence Diagram e dai requisiti UC1/UC9.

Infine, il livello **Test** mostra gli artefatti di verifica che garantiscono la correttezza del refactoring: `AccademicoServiceTest`, `LoginServletTest` e i test associati ai casi d'uso

UC1 e UC9. La loro inclusione nel grafo è giustificata dal fatto che ogni modifica ai service e ai controller coinvolti richiede una revisione dei test di regressione.

Nel complesso, il grafo orizzontale fornisce una visione formale e completa della propagazione del cambiamento, mostrando come il redesign della gerarchia dell'utenza impatti in modo coerente e misurabile tutti i livelli del sistema.

10 Vertical Traceability

10.1 Interpretazione Vertical Traceability Graph

L'*Horizontal Traceability Graph* rappresenta la tracciabilità interna a ciascun livello del ciclo di vita, evidenziando le relazioni tra artefatti omogenei (RAD $\leftrightarrow$ RAD, Design $\leftrightarrow$ Design, Code $\leftrightarrow$ Code, Test $\leftrightarrow$ Test). Tale analisi consente di comprendere come gli elementi di uno stesso layer siano interconnessi e come il cambiamento si propaghi lateralmente all'interno del livello stesso.

Nel livello **RAD**, i requisiti UC1, UC9 e NFR1 risultano correlati tra loro poiché condividono precondizioni, flussi alternativi e vincoli di sicurezza che dipendono dalla struttura dell'utenza. La loro relazione orizzontale giustifica la necessità di trattarli come un blocco coerente nel SIS.

Nel livello di **Design**, l'Object Model della gerarchia utenti è direttamente collegato ai Sequence Diagram SD1, SD7, SD9 e SD10, che ne descrivono il comportamento dinamico. Le relazioni orizzontali tra i diagrammi mostrano come i flussi di autenticazione, recupero email e creazione account condividano la stessa struttura dati e siano quindi simultaneamente impattati dal redesign.

Il livello **Code** è quello maggiormente articolato: i controller interagiscono con i service, i service con i DAO e i DAO con le entità del modello. Le relazioni orizzontali tra i service (`AccademicoService`, `DocenteService`, `StudenteService`, `CoordinatoreService`) derivano dal fatto che essi condividono la stessa gerarchia di partenza e dipendono da strutture dati comuni. Analogamente, i DAO sono correlati tra loro poiché implementano strategie di persistenza simili e condividono la stessa struttura JPA verticalizzata.

Nel livello **Test**, i test di servizio e i test di integrazione sono correlati poiché verificano flussi che condividono componenti comuni. La relazione verticale tra `AccademicoServiceTest` e `LoginServletTest` riflette il fatto che entrambi validano operazioni legate alla gestione dell'utenza e alla corretta propagazione delle eccezioni.

Il grafo verticale, nel suo complesso, permette di identificare cluster di componenti che devono essere trattati come unità coese durante il refactoring. Esso giustifica formalmente la presenza di gruppi di classi nel CIS e fornisce una visione chiara delle dipendenze interne a ciascun layer.

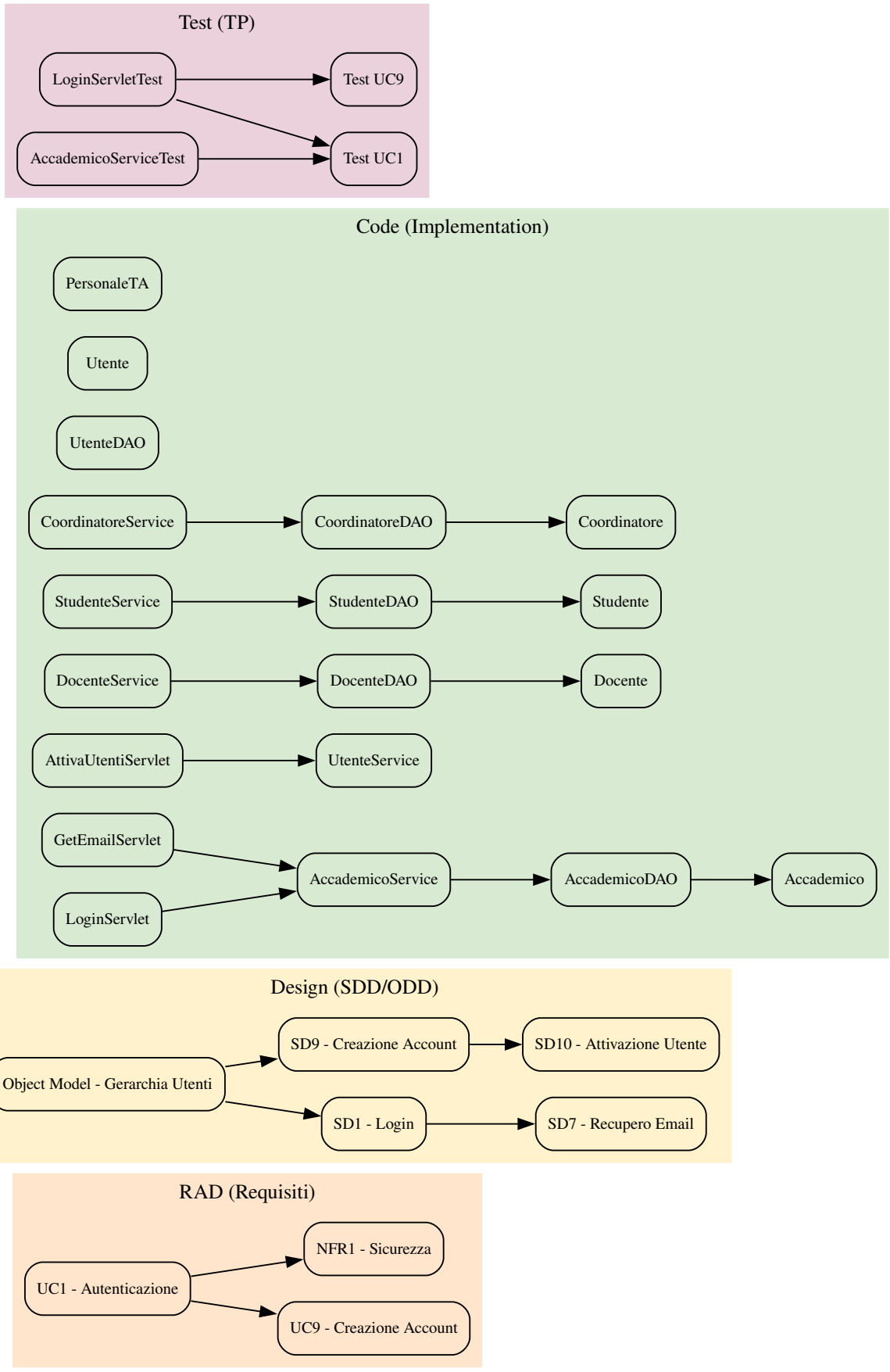


Figura 5: Traceability Graph Verticale tra RAD, Design, Code e Test

11 Traceability Matrix

La gestione della tracciabilità costituisce un elemento fondamentale nel processo di Impact Analysis, poiché consente di collegare in modo sistematico i requisiti, i modelli di design, il codice e i test. In accordo con la letteratura (De Lucia, Oliveto, Fasano), la tracciabilità è modellata attraverso traceability links identificati, tipizzati e dotati di semantica, al fine di supportare la propagazione controllata dell'impatto.

Ogni traceability link è rappresentato come:

$$L = \langle ID,\ source,\ target,\ type,\ direction,\ strength \rangle$$

dove:

- **ID**: identificativo univoco del link (es. `TL-UC1-SD1-REF`);
- **source / target**: artefatti collegati;
- **type**: semantica del legame (refines, implements, calls, tests, derives);
- **direction**: verticale o orizzontale;
- **strength**: forza del legame (*strong*, *medium*, *weak*).

11.1 Tipi di Traceability Link

Tabella 8: Classificazione semantica dei traceability links

Tipo	Descrizione	Esempio UniClass
`refines`	Il design raffina un requisito	SD1 refina UC1
`implements`	Il codice implementa un elemento di design	LoginServlet implements SD1
`calls`	Relazione di invocazione tra componenti	AccademicoService calls AccademicoDAO
`tests`	Un test verifica un componente	LoginServletTest tests LoginServlet
`derives`	Un artefatto deriva informazioni da un altro	Accademico derives Utente

11.2 Traceability Matrix Verticale

La tracciabilità verticale collega artefatti appartenenti a livelli diversi del processo (RAD → Design → Codice → Test).

Tabella 9: Traceability Matrix Verticale

ID Link	Requisito	Design	Codice	Test
TL-UC1-SD1-REF	UC1 – Login	SD1 – Login	LoginServlet.java	LoginServletTest.java
TL-UC9-SD7-REF	UC9 – Creazione Account	SD7 – Recupero Email	GetEmailServlet.java	AccademicoServiceTest.java
TL-NFR1-SD10-REF	NFR1 – Sicurezza	SD10 – Attivazione Utente	AccademicoService.java	Test di integrazione
TL-OBJ-DAO-DER	Object Model	Collapse Accademico	AccademicoDAO.java	–

11.3 Traceability Matrix Orizzontale

La tracciabilità orizzontale collega artefatti appartenenti allo *stesso livello*.

Tabella 10: Traceability Matrix Orizzontale

ID Link	Artefatto A	Artefatto B	Tipo di Relazione
TL-SERV-ACC-CALL	UtenteService	AccademicoService	calls
TL-CTRL-ACC-CALL	LoginServlet	AccademicoService	calls
TL-MODEL-ACC-DER	Accademico	Utente	derives
TL-DAO-ACC-IMP	AccademicoDAO	Accademico	implements persistence
TL-TEST-ACC-TEST	AccademicoServiceTest	AccademicoService	tests

12 Pianificazione Operativa: Approccio Bottom-Up

Al fine di mitigare il *Ripple Effect* (effetto a catena) identificato nell'analisi degli impatti, la fase di implementazione seguirà rigorosamente una strategia **Bottom-Up**. Tale approccio, contrario al flusso di esecuzione logica (dalla UI al DB), è giustificato dalla necessità di stabilizzare le fondazioni del sistema prima di adattare i livelli superiori, garantendo uno stato consistente del codice e la possibilità di eseguire test unitari in isolamento progressivo.

12.1 Razionale della Strategia Bottom-Up

L'analisi delle dipendenze (si vedano i grafi in Appendice A) mostra che i moduli di presentazione (*Controller/JSP*) dipendono fortemente dai Servizi, i quali a loro volta dipendono dalle Entità e dai DAO. Modificare i Controller prima del Modello genererebbe errori di

compilazione bloccanti. Procedendo dal basso verso l'alto (Persistence → Business → Presentation), ogni layer rifattorizzato poggia su un layer sottostante già validato.

12.2 Roadmap di Implementazione

Le attività sono sequenziate in quattro fasi operative distinte:

Fase 1: Layer di Persistenza (Core Stability)

Questa fase mira a ripristinare la compilabilità del modello dati.

- **Azione:** Refactoring delle classi del package `model`. Eliminazione fisica delle classi `Studente.java` e `Docente.java` e introduzione delle annotazioni JPA `@Inheritance` e `@DiscriminatorColumn` su `Utente.java`.
- **Validazione:** Aggiornamento del file `persistence.xml` e verifica tramite `DatabasePopulator`.

Fase 2: Data Access Layer (DAO Consolidation)

Adattamento delle interfacce di accesso ai dati per riflettere la nuova topologia.

- **Azione:** Rimozione dei DAO specifici (`StudenteDAO`, etc.) e potenziamento di `UtenteDAO` per gestire query polimorfiche. Le query specifiche per ruolo verranno incapsulate utilizzando la clausola `WHERE ruolo = ...`.
- **Validazione:** Esecuzione dei Test Unitari sui DAO.

Fase 3: Business Logic Layer (Service & Factory)

Disaccoppiamento della logica di business e introduzione di pattern creazionali.

- **Azione:** Implementazione del pattern *Abstract Factory* (`UtenteFactory`) per centralizzare la logica di creazione complessa delle entità ibride. Refactoring di `UtenteService` per eliminare la dipendenza da `AccademicoService`, riducendo il *God Object anti-pattern*.
- **Validazione:** Esecuzione dei benchmark JMH (es. `UtenteServiceBenchmark.java`) per verificare l'assenza di regressioni prestazionali.

Fase 4: Presentation Layer (Controller & View)

L'ultimo miglio prevede l'adattamento dell'interfaccia utente.

- **Azione:** Refactoring di `LoginServlet` e `AttivaUtentiServlet` per utilizzare esclusivamente `UtenteService`. Revisione delle pagine JSP per gestire l'accesso alle proprietà dinamiche o rilocate (es. matricola).

- **Validazione:** Test di Integrazione (`LoginTest.java`) e User Acceptance Testing (UAT).

13 Strategia di Regression Testing

La strategia di regression testing è stata definita per garantire che il refactoring non introduca regressioni funzionali o comportamentali. Essa prevede una combinazione di test strutturali, funzionali e prestazionali.

In sintesi:

- verifica dell'integrità del modello dati;
- riesecuzione dei test presenti nel CIS;
- test end-to-end sui flussi critici;
- benchmark prestazionali sui servizi principali.

13.1 Mapping CIS – Strategia di Test

La Tabella 11 riassume la relazione tra i componenti del CIS e le attività di testing previste, evidenziandone la priorità.

Componente	**Tipo di Test**	**Priorità**
AccademicoService	Unit + Integration	Alta
UserDirectory	Integration	Alta
LoginServlet	Unit	Alta
JSP Account	Unit Test+Integration	Media
Servizi Conversazioni	Regression funzionale	Media

Tabella 11: Mapping tra CIS e strategia di regression testing

14 Risk Assessment e Mitigazione

Il refactoring comporta alcuni rischi, principalmente legati alla migrazione dei dati, alla compatibilità con i moduli esterni e alla correttezza delle query generate. Tali rischi sono stati mitigati attraverso una combinazione di test, validazioni e controlli di coerenza.

In sintesi:

- i rischi di inconsistenza sono mitigati tramite migrazione controllata;

- le regressioni funzionali sono gestite tramite test approfonditi;
- l'impatto sui moduli esterni è ridotto grazie alla facciata *UserDirectory*.

15 Actual Impact Set and Propagation analysis

15.1 Introduzione

Questo capitolo documenta l'insieme effettivo degli artefatti software modificati a seguito dell'attività di manutenzione sui moduli *Conversazioni*, *Aule/Edifici*, *Gestione Orari* e *Utenti* del sistema UniClass. L'obiettivo è fornire una visione strutturata e tracciabile dell'impatto reale delle modifiche, evidenziando motivazioni, problemi originari e dipendenze coinvolte.

15.2 Propagation Traceability Matrix

Artifact	Change Type	Reason for Change	Original Issue	Impact
Modulo Conversazioni				
`ConversazioniServlet.java`	Modify	Aggiunta inizializzazione delle relazioni LAZY (`autore`, `destinatario`, `topic`) per evitare accessi fuori dal contesto di persistenza.	HTTP 500 causato da LazyInitializationException durante il rendering delle conversazioni.	Caricamento corretto dei messaggi e stabilità del modulo Conversazioni.
`chatServlet.java`	Modify	Correzione gestione attributi request/session e preload delle relazioni LAZY.	NPE e LazyInitializationException durante la visualizzazione della chat.	Chat funzionante e coerente con i dati utente.
`chat.jsp`	Replace	Aggiunti controlli `null` su autore, destinatario e topic; ristrutturazione completa della logica di rendering.	HTTP 500 dovuto a NullPointerException su campi non valorizzati.	Rendering sicuro dei messaggi e robustezza della UI.
`Messaggio.java`	Modify	Aggiunta annotazione `@Table(name="messaggi")` per allineare l'entità allo schema DB previsto.	Errore SQL: `relation "messaggio" does not exist` (disallineamento singolare/plurale).	Corretta mappatura ORM e persistenza dei messaggi.
Modulo Utenti				
`LogoutServlet.java`	Modify	Irrobustimento gestione sessione (`getSession(false)`) e aggiunta gestione eccezioni globale.	HTTP 500 durante il logout per eccezioni non gestite nel redirect o invalidazione.	Logout sicuro e reindirizzamento corretto alla Home.
Modulo Aule ed Edifici				
`EdificioServlet.java`	Replace	Spostamento del caricamento delle lezioni dalla JSP alla servlet; eliminazione accesso a relazioni LAZY; introduzione mappa `lezioniPerAula`.	LazyInitializationException e NPE durante l'accesso alle lezioni in JSP.	Caricamento centralizzato dei dati e separazione corretta tra logica e presentazione.
`edificio.jsp`	Replace	Rimozione uso improprio di `LezioneService` (EJB non iniettato); utilizzo della mappa	NPE: `lezioneDao` null in JSP; errori di rendering.	JSP sicura, priva di logica applicativa e conforme al pattern MVC.

15.3 Conclusioni

L'Actual Impact Set evidenzia come la risoluzione dei problemi abbia richiesto interventi mirati su diversi livelli dell'applicazione. Oltre agli interventi architetturali su *Conversazioni* e *Aule* per il rispetto del pattern MVC, è stato cruciale correggere la configurazione del database (`DatabasePopulator`) e la robustezza dei DAO per evitare crash transazionali. Infine, la rifattorizzazione delle JSP del modulo *Orari* ha garantito la corretta visualizzazione dei dati all'utente finale.

A Backward Dependency Graphs

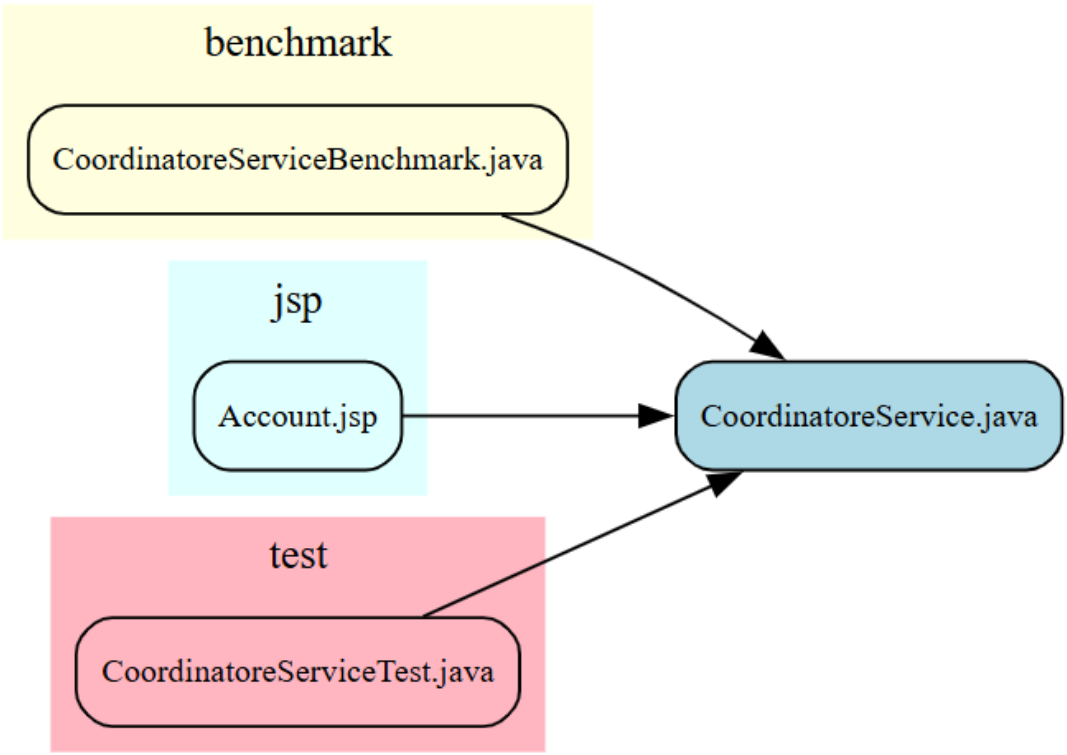


Figura 6: Backward dependency graph di `CoordinatoreService`

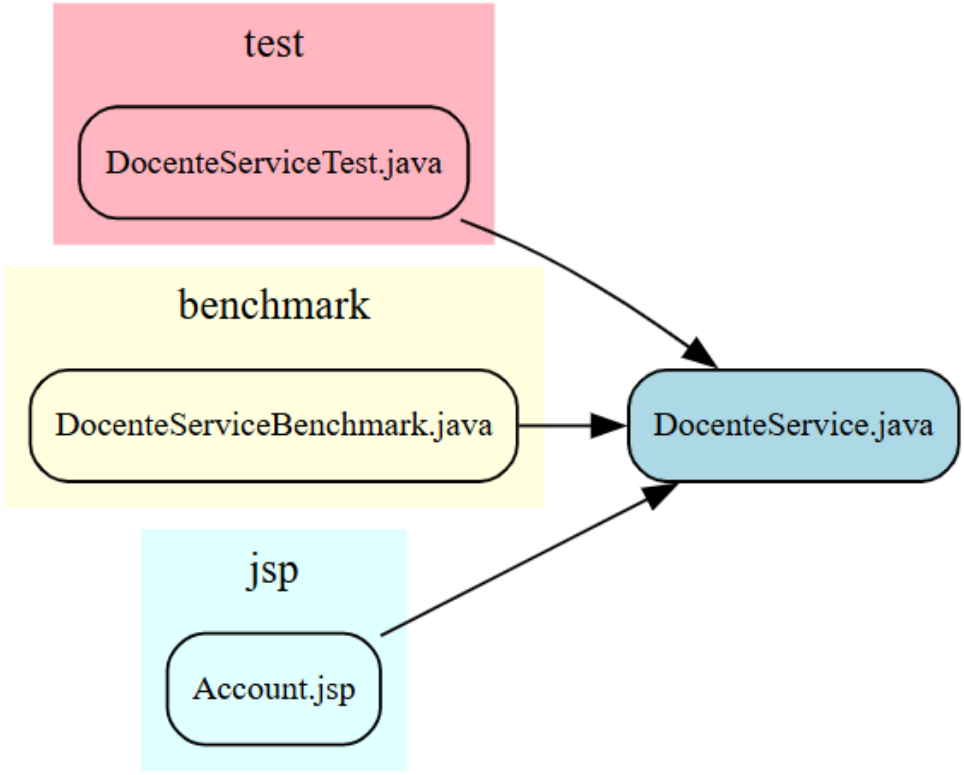


Figura 7: Backward dependency graph di `DocenteService`

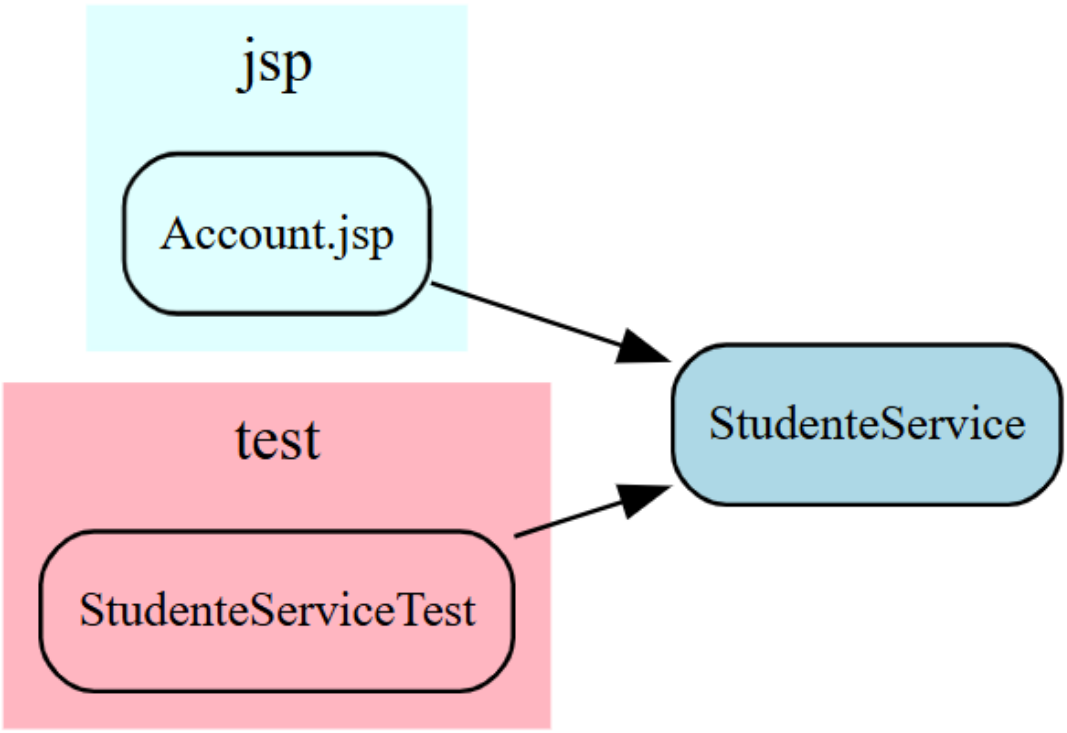


Figura 8: Backward dependency graph di `StudenteService`

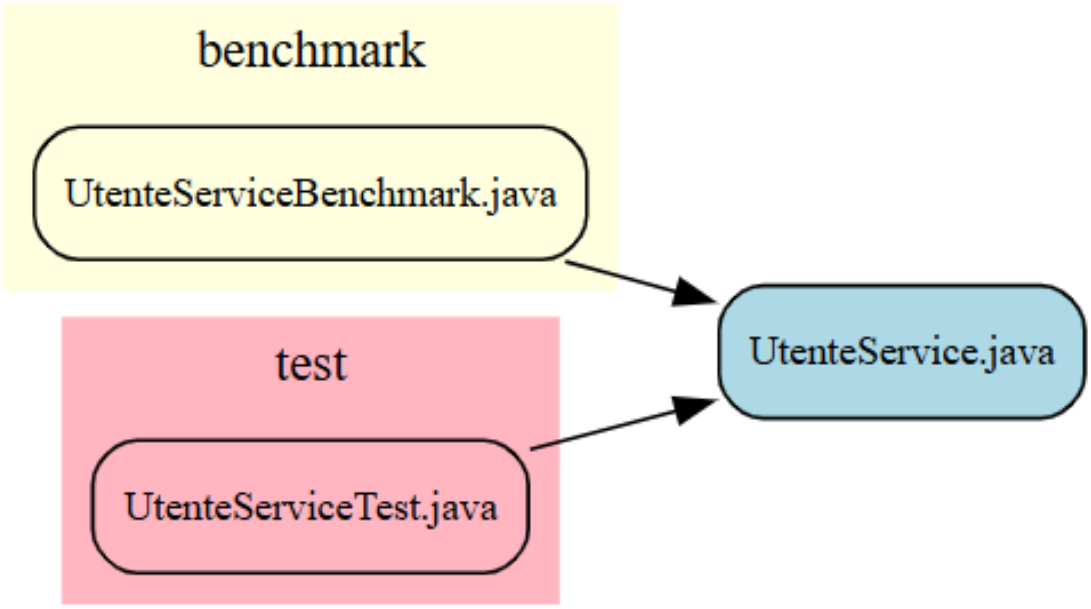


Figura 9: Backward dependency graph di `UtenteService`

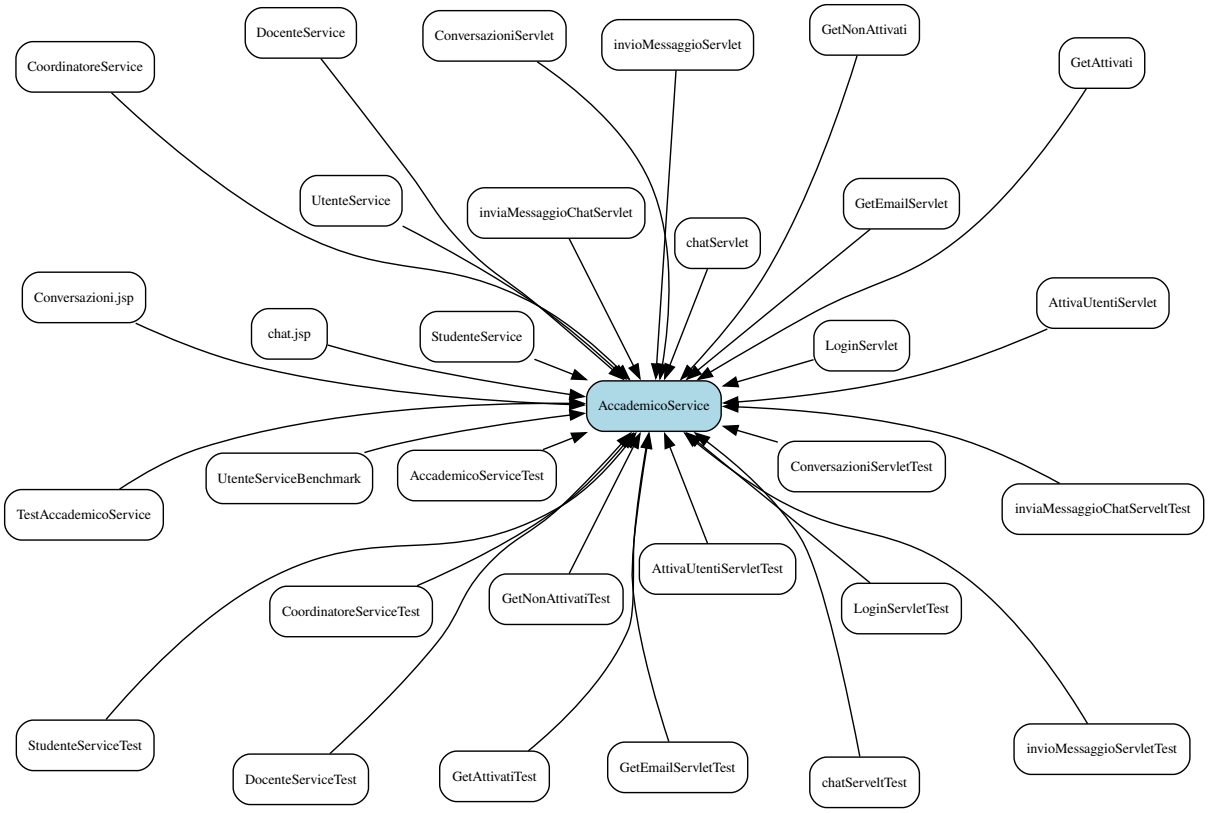


Figura 10: Backward Dependency Graph del God Service `AccademicoService`

B Forward Dependency Graphs

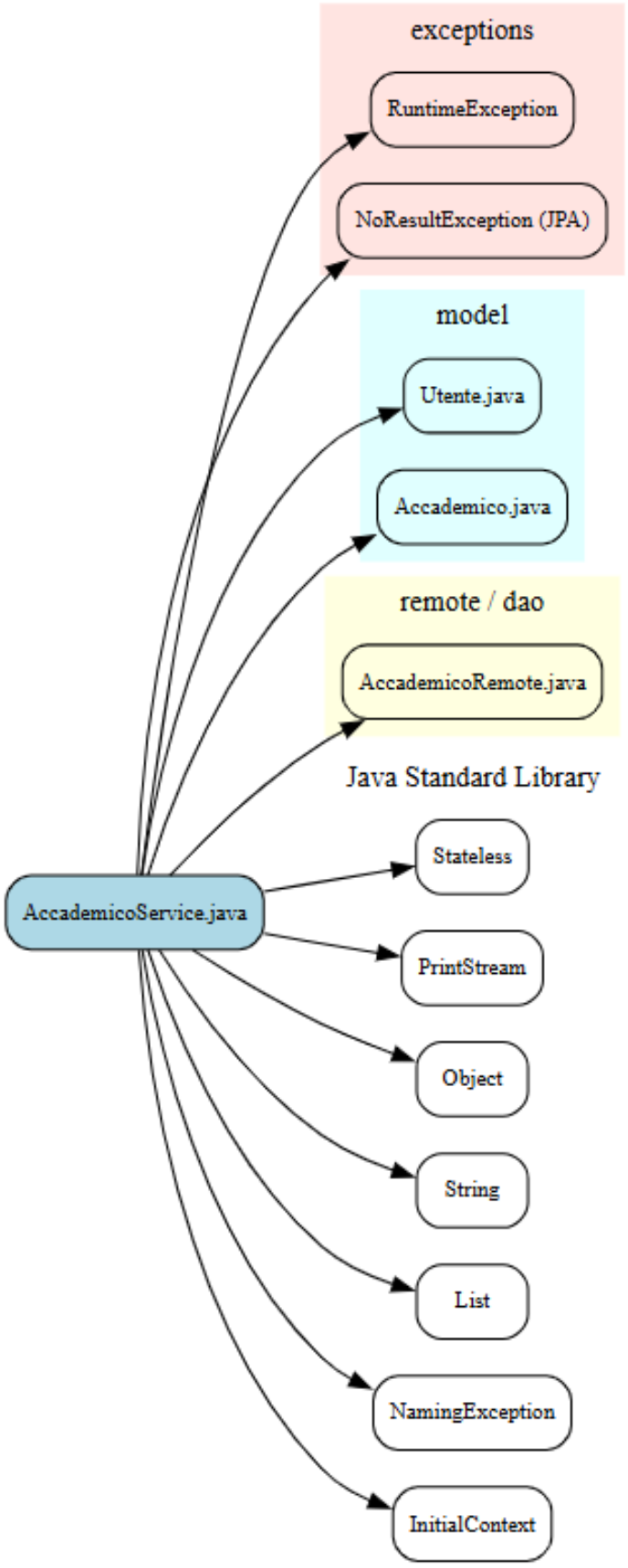


Figura 11: Forward dependency graph di `AccademicoService`

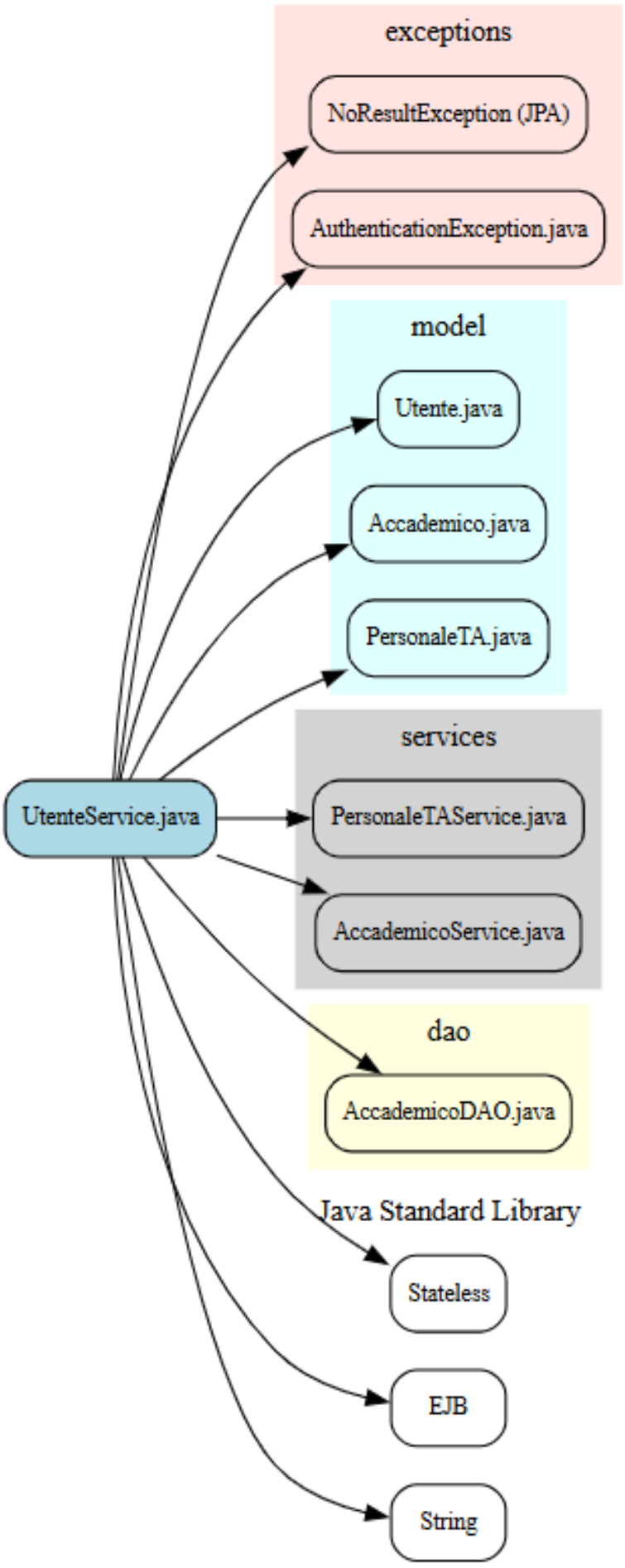
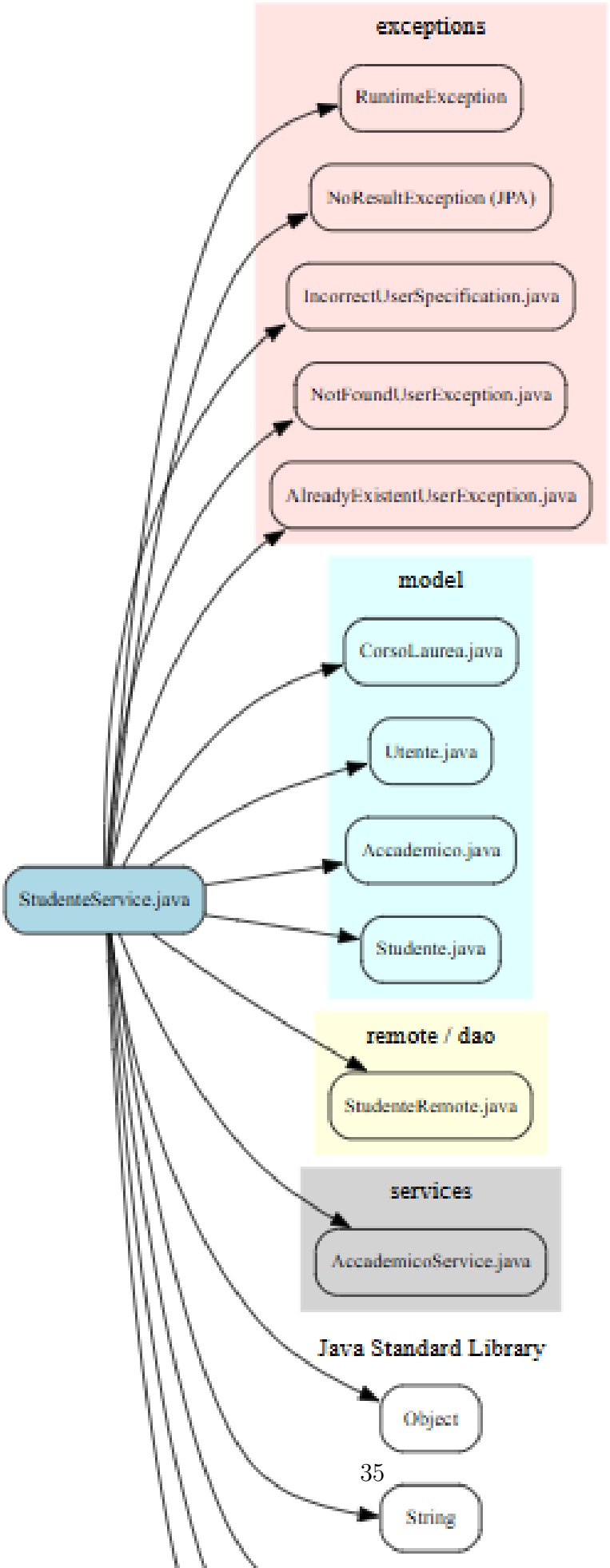


Figura 12: Forward dependency graph di `UtenteService`


exceptions
RuntimeException
NoResultException (JPA)
IncorrectUserSpecification.java
NotFoundUserException.java
AlreadyExistentUserException.java
model
CorsoLaurea.java
Utente.java
Accademico.java
Studente.java
StudenteService.java
remote / dao
StudenteRemote.java
services
AccademicoService.java
Java Standard Library
Object
String

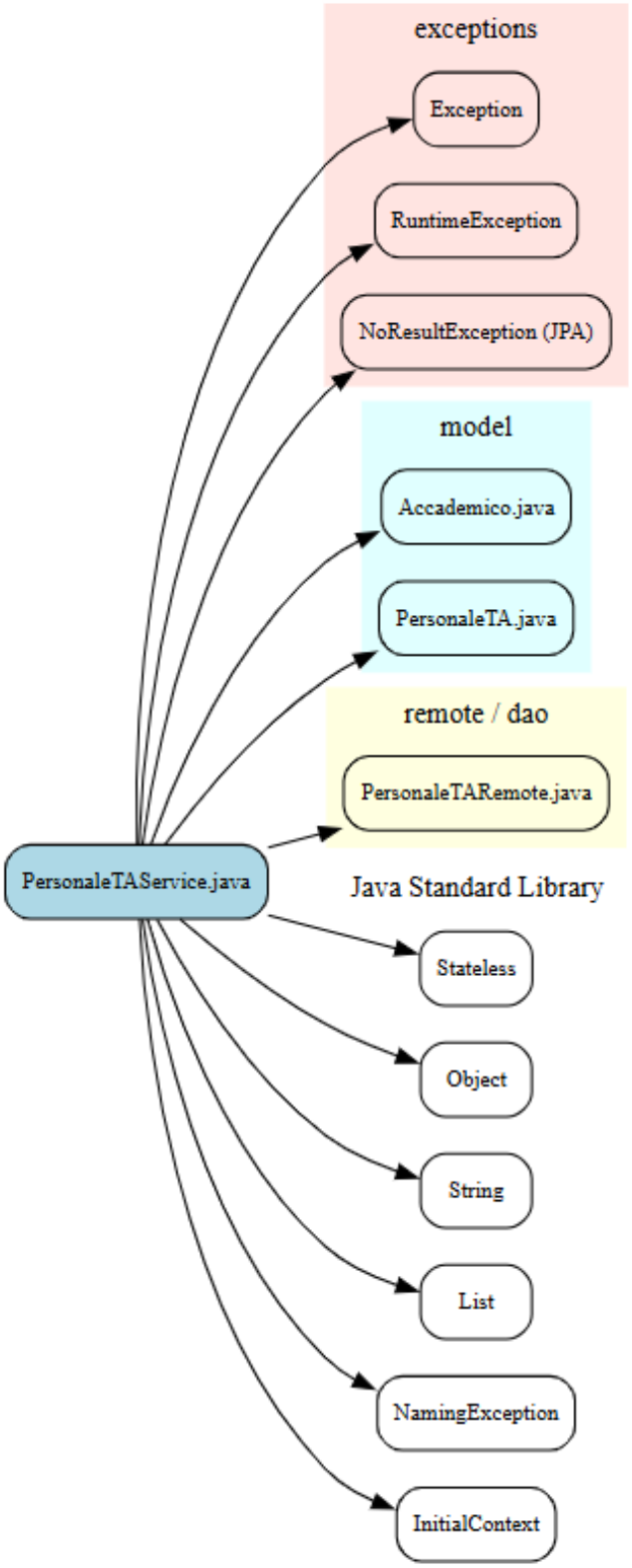


Figura 14: Forward dependency graph di `PersonaleTAService`

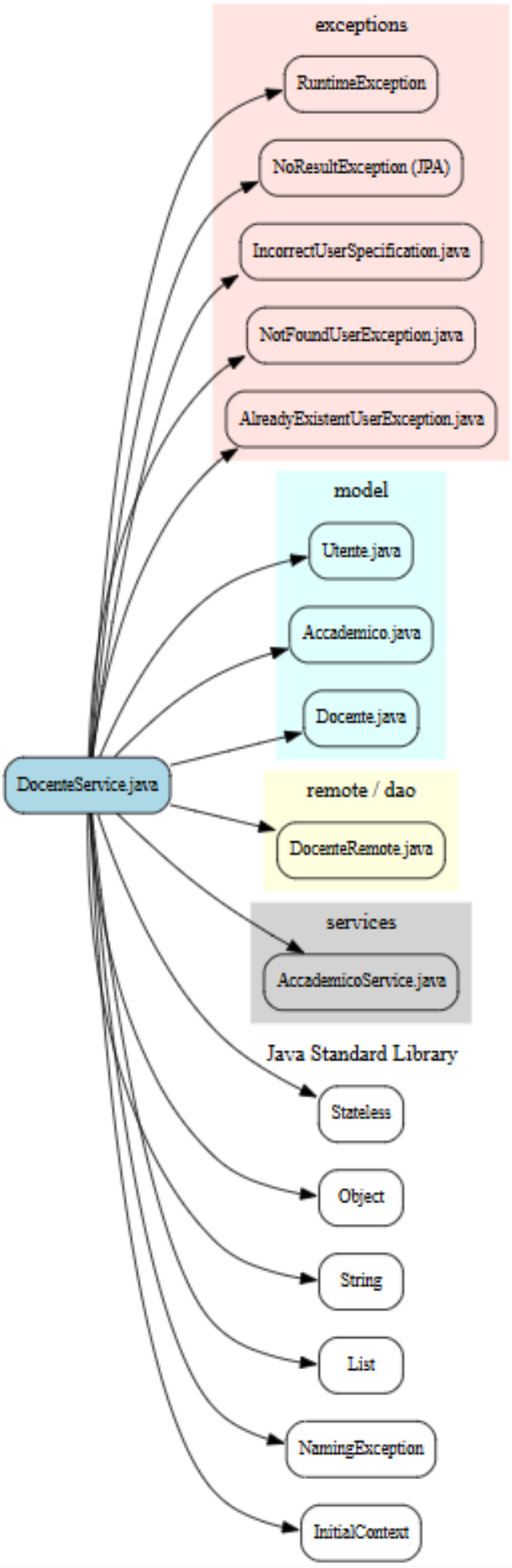


Figura 15: Forward dependency graph di `DocenteService`

C Change Acceptance Criteria

Il refactoring del modulo Utenti è considerato accettato qualora siano soddisfatte le seguenti condizioni:

- tutti i test appartenenti al Primary Impact Set risultano superati;
- i flussi funzionali legacy (login, creazione account, gestione ruoli) risultano invariati dal punto di vista comportamentale;
- il nuovo modello dati è correttamente popolato tramite migrazione controllata;
- i moduli esterni interagiscono esclusivamente tramite *UserDirectory*.

Il soddisfacimento di tali criteri garantisce che il cambiamento introdotto rappresenti un'evoluzione sicura e controllata del sistema.