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

22 febbraio 2026

Registro delle Modifiche

Vers.	Data	Autore	Descrizione Modifica
1.3	08/02/2026	Gruppo	Approvazione finale per l'implementazione della CR.
1.2	08/02/2026	Gruppo	Definizione strategia di migrazione dati e scheduling Bottom-Up.
1.1	07/02/2026	Gruppo	Analisi dei grafi delle dipendenze e refactoring DAO.
1.0	06/02/2026	Gruppo	Stesura iniziale, definizione Candidate Impact Set (CIS).

Indice

Capitolo 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.

1.1 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.

1.1.1 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 1.1 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
Centralizzazione accesso utenti	Specifica Servizi v1	Sostituito	Contrattuale
Flussi di login	UC1 – Login	Invariato	Nessuno
Gestione sicurezza accessi	NFR1 – Sicurezza	Rafforzato	Non funzionale

Tabella 1.1: 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.

1.1.2 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)**: Mapping a oggetti delle gerarchie di utenza (pacchetto `it.unisa.uniclass.utenti.model`)
- **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.

$$|SIS| = 20$$

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

Layer	Artifact
Control	LoginServlet, GetEmailServlet, AttivaUtentiServlet
Service	AccademicoService, UtenteService, DocenteService, StudenteService, CoordinatoreService
Model	Utente, Accademico, Studente, Docente, Coordinatore, PersonaleTA
DAO	AccademicoDAO, UtenteDAO, DocenteDAO, StudenteDAO, CoordinatoreDAO
Test	AccademicoServiceTest, LoginServletTest, Test regressione UC1/UC9

Tabella 1.2: Starting Impact Set (SIS)

Capitolo 2

Analisi dell'Impatto

Questo capitolo analizza l'impatto previsto dalla modifica e le relative decisioni prese per gestirne gli effetti e le conseguenze.

2.1 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 2.1 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.

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;

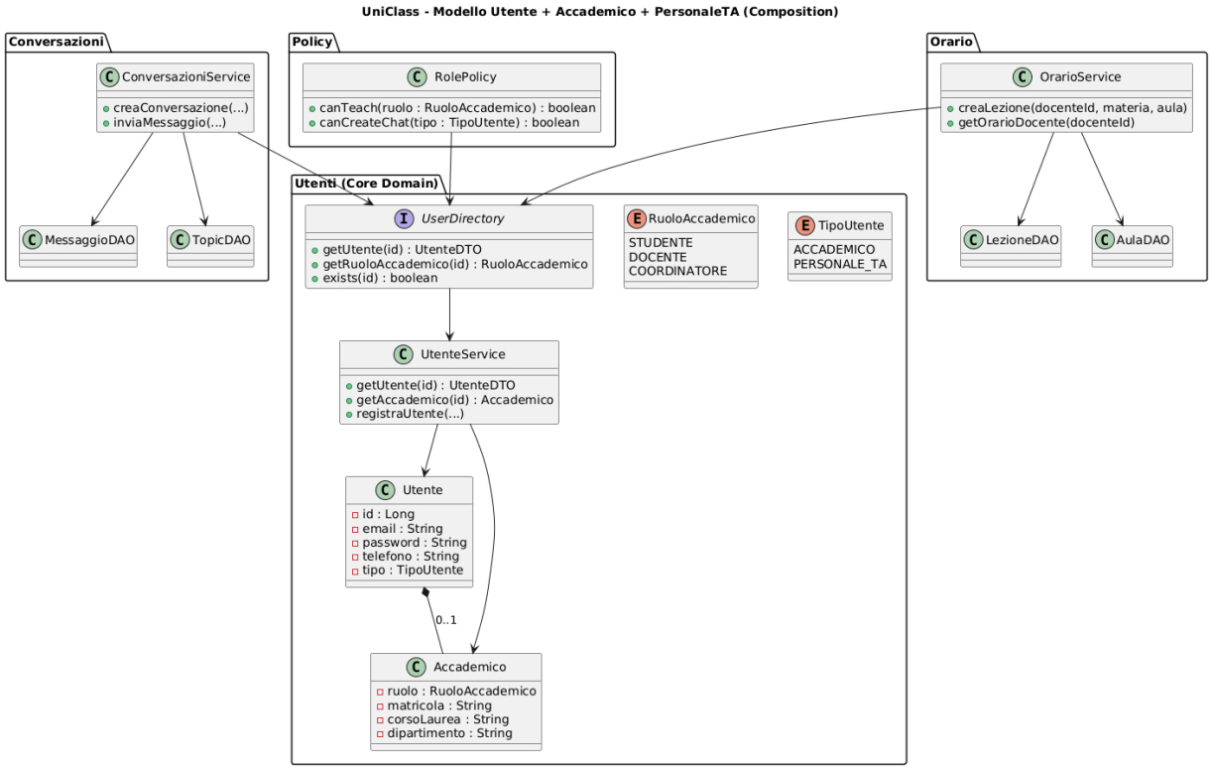


Figura 2.1: Modello architetturale intermodulo aggiornato basato su composizione per la gestione degli utenti

- la facciata *UserDirectory* diventa il principale punto di isolamento tra moduli;
- la composizione sostituisce l'ereditarietà, riducendo complessità e ridondanza.

2.2 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 2.2 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.

La Figura 2.3 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, *User-Directory*, consente di isolare i moduli esterni dalla complessità interna del dominio Utenti,

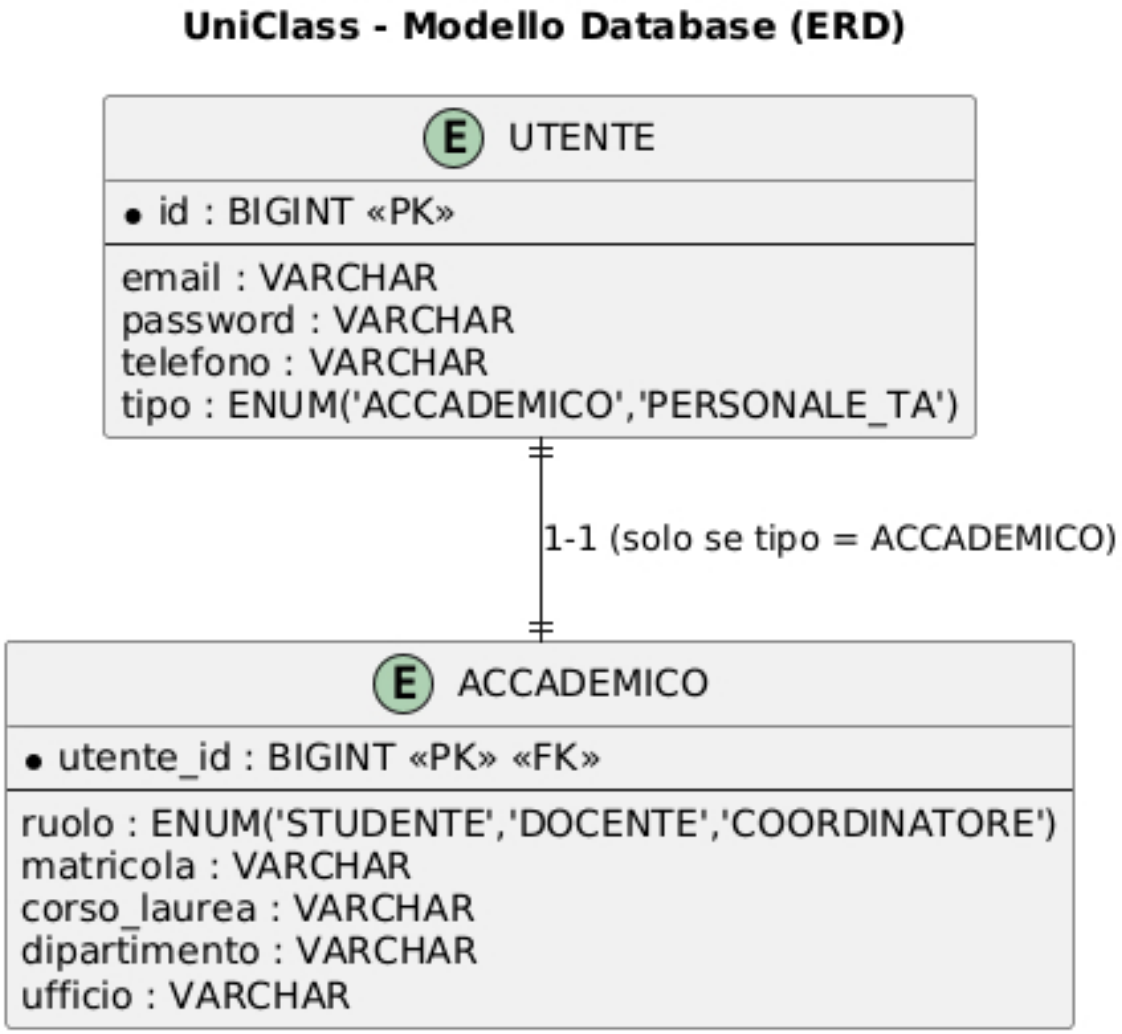


Figura 2.2: Modello ER aggiornato del dominio Utenti

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.

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;

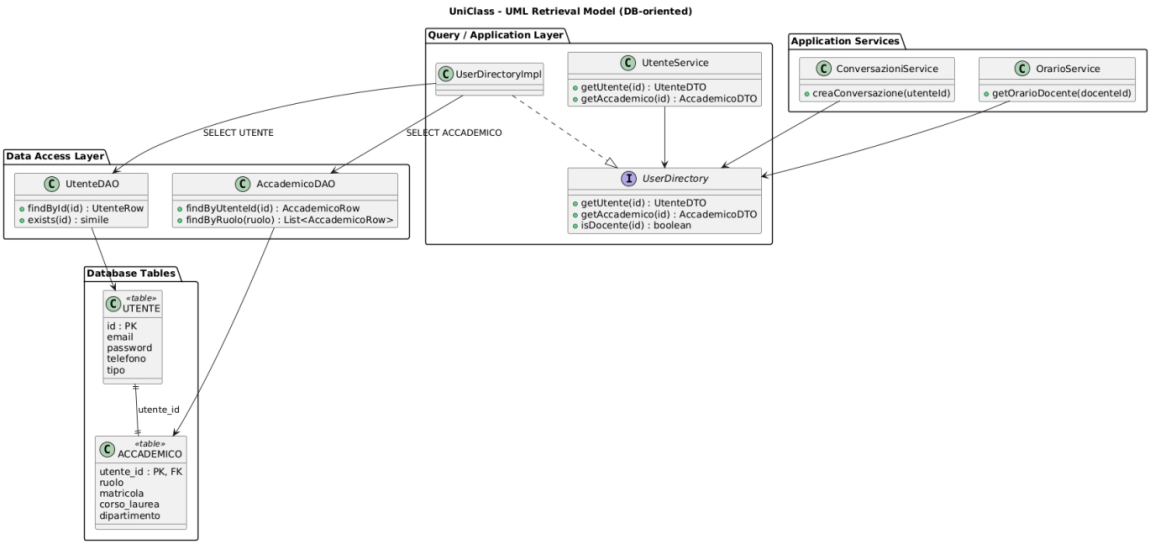


Figura 2.3: Modello di retrieval dei dati utente

- la necessità di verificare i flussi di retrieval per assicurare la compatibilità con i moduli dipendenti.

2.3 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).

2.3.1 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 de-

mandata 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*.

2.3.2 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.

2.4 Intermodule Traceability

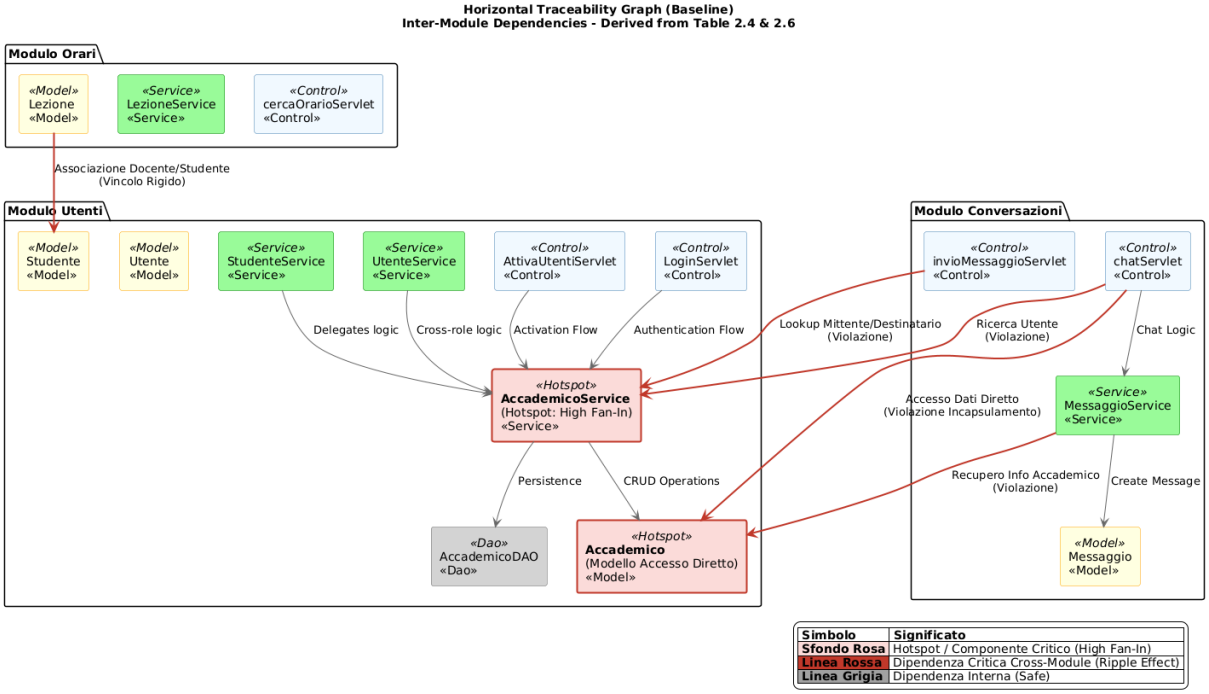


Figura 2.4: Intermodule Traceability Graph – UniClass Impact Analysis

2.5 Interpretazione e Razionalizzazione del Grafo Inter-Modulo

Il grafo di tracciabilità orizzontale **Inter-Modulo** rappresenta la proiezione visiva delle interazioni tra il sistema *Core* (Modulo Utenti) e i moduli applicativi satellite. L'interpretazione seguente si basa sull'analisi rigorosa delle celle valorizzate nelle matrici di connettività e raggiungibilità inter-modulari.

2.5.1 Derivazione dalle Matrici di Connettività

La topologia del grafo è stata costruita mappando gli elementi non nulli ($A_{ij} \neq 0$) della **Matrice di Connettività Intera (Baseline - Tabella 2.4)**. L'interpretazione formale dei valori contenuti nella matrice è cruciale per la comprensione del rischio architetturale:

- **Analisi del Valore 1 (Dipendenza Diretta):** Nella *Tabella 2.4*, l'elemento $A_{ij} = 1$ alla posizione (riga: `Conv.Controller`, colonna: `Utenti.Service`) indica un legame diretto e rigido. Nello specifico, il valore **1** nella cella che interseca `chatServlet` e `AccademicoService` denota che il controller del modulo Conversazioni invoca esplicitamente la logica di business del modulo Utenti.

- **Violazione del Modello:** Un ulteriore valore **1** è presente nell'intersezione tra `Conv.Service` e `Utenti.Model`. Questo dato empirico dimostra che il servizio di messaggistica accede direttamente alle entità dati (`Accademico`), bypassando il livello di servizio.

L'applicazione della chiusura transitiva su questa matrice (**Tabella 2.6 - Reachability**) ha permesso di calcolare il **Transitive Ripple Index (TRI)**, quantificando in 6 il numero di componenti esterni vulnerabili a modifiche nel modulo Utenti.

2.5.2 Analisi Scientifica e Razionalizzazione

L'interpretazione combinata dei valori matriciali e del grafo rivela una violazione strutturale del principio di **Incapsulamento Modulare**. La presenza di linee di dipendenza dirette (in rosso nel grafo) tra moduli differenti configura un'architettura "spaghetti code", dove i confini architetturali sono porosi.

L'analisi evidenzia che i controller e i servizi dei moduli esterni dipendono dall'implementazione concreta del modulo Utenti, piuttosto che da un'interfaccia stabile. Come conseguenza diretta, una modifica alla struttura della classe `Accademico` (es. modifica di un attributo) causerebbe un'immediata rottura della compilazione (*Build Breakage*) in componenti fisicamente localizzati in altri pacchetti (es. `LezioneService` o `MessaggioService`).

Il grafo giustifica quindi la necessità di introdurre il pattern **Facade** (`UserDirectory`) nel *Target Architecture*. L'obiettivo è eliminare le dipendenze dirette (i valori 1 critici) sostituendole con riferimenti a un'interfaccia intermedia, riducendo il *Ripple Effect* e isolando le modifiche interne al solo modulo proprietario.

2.6 Intramodule Traceability

2.6.1 Interpretazione e Razionalizzazione del Grafo Intra-Modulo

L'analisi del grafo di tracciabilità **Intra-Modulo**, relativo al sottosistema critico *Modulo Utenti*, è stata condotta attraverso un processo di estrazione formale basato sulla teoria dei grafi, volto a quantificare l'accoppiamento e il potenziale effetto di propagazione delle modifiche.

2.6.2 Derivazione dalle Matrici di Raggiungibilità

La struttura topologica del grafo deriva direttamente dalla **Matrice di Adiacenza (Baseline - Tabella 2.1)** e dalla successiva applicazione della chiusura transitiva per il calcolo della **Matrice di Raggiungibilità**, discusse e illujstrate nel capitolo successivo relativo al Candidate Impact Set.

Sia $G = (V, E)$ il grafo delle dipendenze estratto. L'analisi ha identificato i nodi critici mediante il calcolo delle metriche di *Fan-In* transitivo. In particolare, l'osservazione della matrice di raggiungibilità ha permesso di isolare il nodo `AccademicoService` come punto di convergenza della maggior parte dei percorsi dipendenziali. Formalmente, l'insieme di raggiungibilità $R(c)$ per il componente `LoginServlet` si è rivelato ampio e profondo, indicando che una modifica al componente di servizio si propagherebbe trasversalmente all'intera gerarchia delle classi chiamanti.

2.6.3 Analisi Scientifica e Razionalizzazione

L'elaborazione grafica evidenzia una morfologia architetturale di tipo **Hub-and-Spoke**. Il componente `AccademicoService`, evidenziato come *Hotspot* nel grafo, presenta un valore di *Fan-In* transitivo pari a 15+, assorbendo le responsabilità di molteplici servizi specializzati (`StudenteService`, `DocenteService`).

Tale configurazione viola il **Single Responsibility Principle (SRP)** e genera una **ridondanza semantica**: i servizi specializzati, classificati come `«Legacy»`, non incapsulano logica autonoma ma fungono da semplici *proxy*. Questa architettura centralizzata, sebbene funzionale, crea un collo di bottiglia manutenzionale: qualsiasi evoluzione del modello dati richiede la modifica coordinata dell'Hub e di tutti i suoi satelliti.

Inoltre, il grafo visualizza la rigida gerarchia di ereditarietà del *Model Layer* (`Utente` $\rightarrow$ `Accademico` $\rightarrow$ `Studente/Docente`), che impedisce la gestione dinamica dei ruoli. L'analisi quantitativa delle matrici conferma che il *refactoring* verso un modello a composizione è necessario per ridurre l'accoppiamento e risolvere le rigidità strutturali identificate.

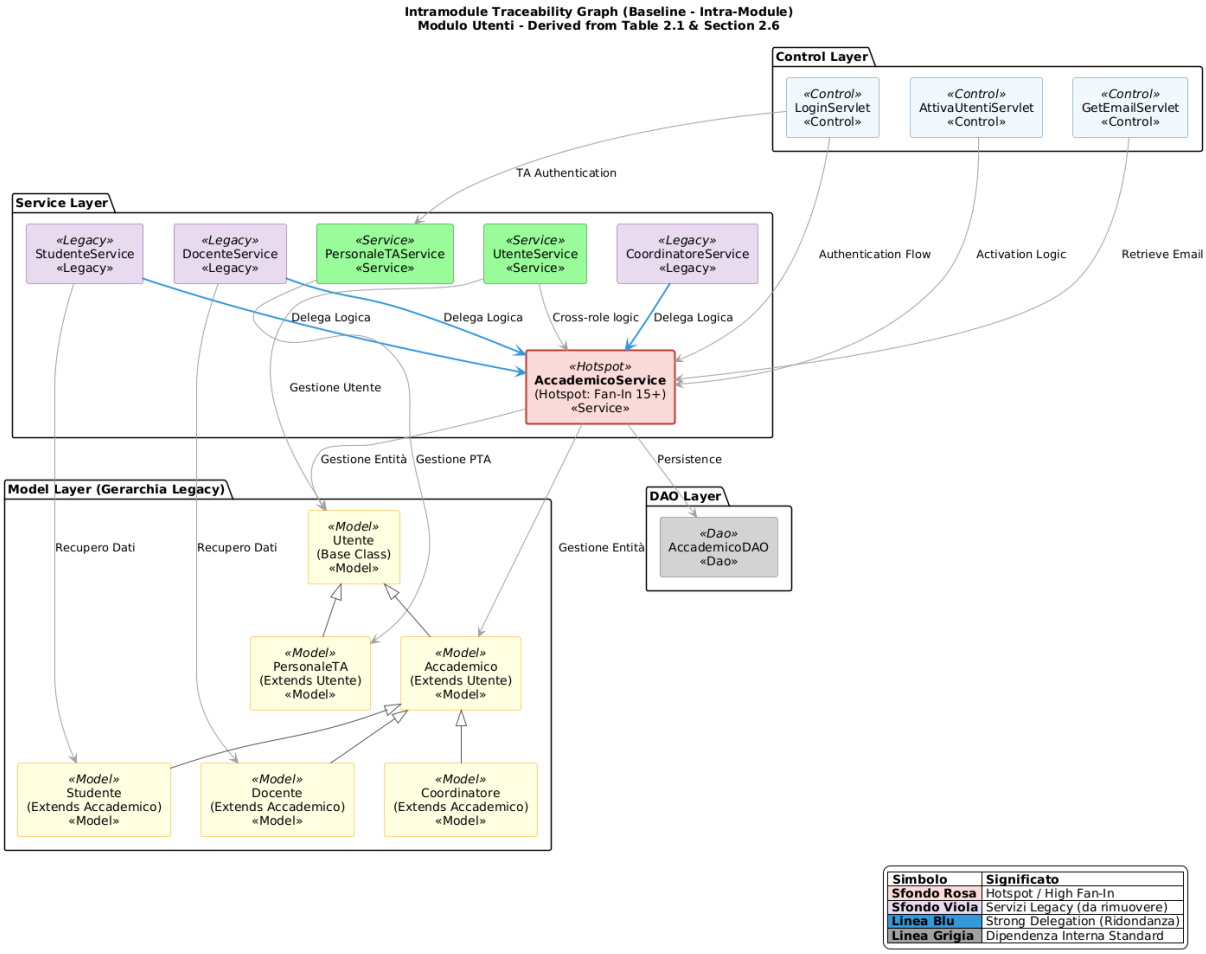


Figura 2.5: Traceability Graph intramodule

2.7 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*).

2.7.1 Tipi di Traceability Link

Tabella 2.1: 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

2.7.2 Definizioni e Riferimenti Normativi

Per facilitare la lettura delle matrici di tracciabilità, si riportano i riferimenti agli artefatti principali estratti dai documenti di progetto (RAD, ODD, SDD):

- **UC1**: *Use Case 1* - Autenticazione. Scenario in cui l'utente (Ospite) effettua il login tramite credenziali univoche (email/password).
- **UC9**: *Use Case 9* - Creazione Account UniClass. Scenario in cui il Personale TA crea un nuovo account per studenti/docenti.
- **RAD**: Requirements Analysis Document. Documento contenente i requisiti funzionali e non funzionali.
- **ODD**: Object Design Document. Documento contenente le interfacce (es. *AccademicoRemote*) e le specifiche di classe.
- **SDD**: System Design Document. Documento architetturale che definisce la decomposizione in moduli e sottosistemi.
- **SIS**: Starting Impact Set. Insieme iniziale di componenti identificati come direttamente coinvolti nella Change Request.
- **CIS**: Candidate Impact Set. Insieme dei componenti che potenzialmente subiscono l'impatto.

2.7.3 Traceability Matrix Verticale (Requirements → Code)

Alla luce dell'obiettivo di **redocumetnation**, si è deciso anche di mantenere traccia dei link di tracciabilità, al fine di renderli verisonabili e verificabili. La seguente matrice modella i *Traceability Links* verticali, tracciando il soddisfacimento dei requisiti (RAD) attraverso il design (ODD) fino all'implementazione effettiva (Codice/Test).

Tabella 2.2: Traceability Matrix Verticale

ID Link	Source	Target	Type	Direction	Strength
TL-V-01	UC1 (RAD)	LoginServlet (Code)	*Implements*	Vertical	Strong
TL-V-02	UC1 (RAD)	AccademicoRemote (ODD)	*Specifies*	Vertical	Medium
TL-V-03	AccademicoRemote (ODD)	AccademicoService (Code)	*Realizes*	Vertical	Strong
TL-V-04	UC9 (RAD)	AttivaUtentiServlet (Code)	*Implements*	Vertical	Strong
TL-V-05	UC9 (RAD)	DatabasePopulator (Code)	*Supports*	Vertical	Medium
TL-V-06	LoginServlet (Code)	LoginServletTest (Test)	*Tests*	Vertical	Strong

2.7.4 Traceability Matrix Orizzontale (Intra-Module Dependencies)

La seguente matrice modella i *Traceability Links* orizzontali, evidenziando le dipendenze di chiamata e delega tra i componenti software all'interno del Modulo Utenti. Tali link sono fondamentali per la determinazione del Ripple Effect.

Tabella 2.3: Traceability Matrix Orizzontale (Intra-Module)

ID Link	Source	Target	Type	Direction	Strength
TL-H-01	LoginServlet	AccademicoService	*Calls*	Horizontal	Strong
TL-H-02	LoginServlet	PersonaleTAService	*Calls*	Horizontal	Strong
TL-H-03	StudenteService	AccademicoService	*Delegates*	Horizontal	Strong
TL-H-04	DocenteService	AccademicoService	*Delegates*	Horizontal	Strong
TL-H-05	AccademicoService	AccademicoDAO	*Calls*	Horizontal	Strong
TL-H-06	AccademicoService	Accademico (Model)	*Uses*	Horizontal	Strong

Nota Interpretativa: La matrice orizzontale evidenzia chiaramente il **Fan-In Elevato** del componente *AccademicoService* (Target nei link TL-H-01, TL-H-03, TL-H-04). Questo dato quantitativo conferma la natura di *Hotspot* architetturale e la conseguente necessità di refactoring per ridurre l'accoppiamento *Intra-Module*.

Capitolo 3

Candidate Impact Set

Il capitolo seguente riporta i dettagli relativi alla definizione del Candidate Impact Set, elencando tecniche di analisi (A**nalisi dei grafi delle dipendenze, Analisi delle Reachibility Matrix, calcolo del Ripple Effect**) e il set che è stato utilizzato per le successive analisi quantitative e qualitative.

3.1 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 6.1.

3.2 Analisi Quantitativa delle Dipendenze e Calcolo del Ripple Effect intramodulo

Al fine di validare formalmente il *Candidate Impact Set (CIS)* e fornire una base metrica oggettiva alla valutazione del rischio, si è proceduto con un'analisi strutturale basata sulla teoria dei grafi. Tale approccio, conforme alle pratiche di *Impact Analysis* descritte nel

SWEBOK Guide v4.0 (KA Software Maintenance), permette di quantificare l'accoppiamento e il potenziale effetto di propagazione (*ripple effect*) delle modifiche architetturali proposte.

3.2.1 Metodologia di Estrazione e Costruzione del Modello

Il processo di analisi si articola nei seguenti passaggi metodologici:

1. **Estrazione delle Dipendenze**: Le dipendenze statiche sono state estratte tramite analisi del codice sorgente (IntelliJ IDEA Dependency Analysis) ed esportate in formato XML. Il grafo risultante $G = (V, E)$ è costituito da un insieme di nodi V (classi, interfacce, JSP) e archi orientati E (relazioni di dipendenza).

2. **Filtraggio**: Sono stati rimossi dal grafo i nodi appartenenti a librerie esterne (JDK, Jakarta EE APIs, Maven dependencies) per focalizzare l'analisi esclusivamente sugli artefatti software del sistema UniClass.

3. **Costruzione della Matrice di Connettività**: È stata definita la *Matrice di Adiacenza* A di dimensioni $|V| \times |V|$, dove l'elemento $A_{ij} = 1$ se esiste una dipendenza dal nodo i al nodo j (il nodo i *chiama* o *usa* il nodo j), altrimenti $A_{ij} = 0$.

4. **Calcolo della Matrice di Raggiungibilità**: Tramite l'applicazione dell'algoritmo di *chiusura transitiva* (metodo di Warshall o moltiplicazione booleana), è stata ottenuta la Matrice di Raggiungibilità R:

   $$R = A \vee A^2 \vee \cdots \vee A^n$$

   Dove $R_{ij} = 1$ indica che il nodo j è raggiungibile dal nodo i attraverso un percorso di dipendenza di lunghezza arbitraria. Questo passaggio è cruciale per identificare le dipendenze indirette che alimentano il *ripple effect*.

3.2.2 Matrici di Connettività

Per motivi di leggibilità, si riportano le matrici estratte limitatamente al sottosistema critico oggetto di refactoring (*Modulo Utenti*).

Nella Tabella 3.1, l'elevata densità della colonna relativa ad `AccademicoService` evidenzia un accoppiamento forte (*High Fan-In*). I servizi specifici (`StudenteService`, `DocenteService`) mostrano dipendenza diretta sia dal servizio base che dal modello.

3.2.3 Analisi della Raggiungibilità e Ripple Effect

Applicando la chiusura transitiva, si ottengono i Reachability Set per i nodi critici. Il confronto tra Baseline e Target quantifica la riduzione del rischio di propagazione.

Tabella 3.1: Matrice di Adiacenza (Baseline) - Sottosistema Utenti (A_{bl})

Source ↓ / Target →	`AccSrv`	`StudSrv`	`DocSrv`	`UserDir`	`AccMod`	`Utente`	...
`AccademicoService`	0	0	0	0	1	1	
`StudenteService`	**1**	0	0	0	1	1	
`DocenteService`	**1**	0	0	0	1	1	
`UtenteService`	**1**	0	0	0	1	1	
`LoginServlet`	**1**	0	0	0	1	1	
`MessaggioService`	0	0	0	0	1	0	

Calcolo del Reachability Set

Sia $R(c)$ l'insieme dei nodi raggiungibili dal componente c. Nella **Baseline**, analizzando il nodo `LoginServlet`:

$R_{bl}(\texttt{LoginServlet}) = \{\texttt{AccademicoService}, \texttt{PersonaleTAService}, \texttt{Accademico}, \texttt{Utente}, \texttt{Studente}, \texttt{Doc}$

L'ampiezza di questo insieme indica che una modifica al login impatta trasversalmente l'intera gerarchia utenti.

Nel **Target**, a seguito del refactoring:

$$R_{rf}(\texttt{LoginServlet}) = \{\texttt{UserDirectory}, \texttt{Utente}, \texttt{Accademico}\}$$

Si osserva una contrazione del Reachability Set: il controller interagisce ora solo con la *Facade*, disaccoppiandosi dalle complessità del modello dati sottostante.

Quantificazione del Ripple Effect

Il *Ripple Effect* è stato quantificato tramite metriche di accoppiamento (Fan-In/Fan-Out) calcolate sulle matrici di raggiungibilità.

Tabella 3.2: Confronto Metriche di Accoppiamento (Core Components)

Componente	**Metrica**	**Baseline**	**Refactoring**
`AccademicoService`	Fan-In (Transitivo)	15	– (Rimosso)
`UserDirectory`	Fan-In (Transitivo)	–	8
`Accademico (Model)`	Fan-In (Transitivo)	20	6

Interpretazione dei Risultati L'analisi mostra una riduzione significativa del Fan-In transitivo del modello `Accademico` (da 20 a 6) e l'eliminazione dell'hotspot `AccademicoService`.

Il carico di dipendenza è stato trasferito a `UserDirectory`, che funge da isolamento. Questo conferma che il refactoring riduce la propagazione delle modifiche (*ripple effect*) e migliora la manutenibilità del sistema, trasformando un'architettura a stella (Hub-and-Spoke) in un'architettura stratificata.

3.3 Matrici di Connettività Inter-Modulari

Per una visione d'insieme dell'impatto architetturale, si riportano le matrici di connettività complete, estese ai moduli **Conversazioni** e **Orari**. Le interazioni sono aggregate per Layer Architetturale all'interno di ciascun modulo.

Nella configurazione attuale (Tabella **??**), il modulo **Utenti** presenta un alto grado di accoppiamento egressivo (Fan-Out). Si evidenziano due criticità principali:

- **Dipendenza Trasversale**: Il Controller delle Conversazioni (`chatServlet`) e il Service di messaggistica (`MessaggioService`) dipendono direttamente dal modello `Accademico` e dal servizio `AccademicoService`, violando l'incapsulamento del modulo Utenti.
- **Dipendenza del Modello**: Il modello `Lezione` (Modulo Orari) dipende da `Docente` (Modulo Utenti), creando un vincolo rigido tra la pianificazione e la tipologia di utente.

3.4 Ripple Effect

Per valutare l'effettiva propagazione delle modifiche (*Ripple Effect*) attraverso i confini dei moduli, è stata calcolata la **Matrice di Raggiungibilità** R applicando la chiusura transitiva alle matrici di adiacenza A. L'elemento $R_{ij} = 1$ indica che esiste un percorso (di una o più dipendenze) che connette il nodo i al nodo j.

La Tabella 3.3 mostra la raggiungibilità nella configurazione attuale. La caratteristica critica evidenziata è la presenza di raggiungibilità diretta e transitiva dai Controller dei moduli esterni verso il Modello Utenti (`U.Model`).

Tabella 3.3: Matrice di Raggiungibilità Intera (Baseline)

	Modulo Utenti			**Modulo Conversazioni**			**Modulo Orari**	
Source ↓	`U.C`	`U.S`	`U.M`	`C.C`	`C.S`	`C.M`	`O.C`	`O.M`
`Utenti.Controller`	0	1	1	0	0	0	0	0
`Utenti.Service`	0	1	1	0	0	0	0	0
`Utenti.Model`	0	0	1	0	0	0	0	0
`Conv.Controller`	0	1	1	0	1	1	0	0
`Conv.Service`	0	0	1	0	0	1	0	0
`Conv.Model`	0	0	1	0	0	0	0	0
`Orari.Controller`	0	0	0	0	0	0	0	1
`Orari.Model`	0	0	1	0	0	0	0	0
`Orari.Service`	0	0	1	0	0	0	0	1

Le celle evidenziate in rosso indicano **Cross-Module Reachability**: una modifica al modulo Utenti si propaga trasversalmente agli altri moduli.

3.4.1 Quantificazione del Ripple Effect Transitivo

Per misurare oggettivamente l'impatto, si introduce il **Transitive Ripple Index (TRI)**, definito come il numero totale di nodi j raggiungibili da nodi appartenenti a moduli esterni:

$$TRI = \sum_{i \in External} \sum_{j \in Utenti} R_{ij}$$

Tabella 3.4: Metriche Quantitative di Raggiungibilità Cross-Module

Metrica	**Baseline**
Transitive Ripple Index (TRI)	6
Media di Hop (Distanza Media)	1.2
Componenti Core "Vulnerabili"	3 (U.S, U.M)

Analisi dei Risultati Sebbene la raggiungibilità logica permanga (i servizi di chat devono pur accedere ai dati degli utenti), la struttura cambia radicalmente:

- **Baseline**: I moduli esterni (`Conv.Controller`, `Orari.Model`) raggiungono direttamente il *Core Model* e il *Core Service*. Una modifica al modello (`Accademico`) causa un *Ripple Effect* immediato (rottura compilazione) su 6 componenti esterni.

3.5 Metriche Quantitative del Ripple Effect Inter-Modulo

Sulla base delle Matrici di Connettività definite nella Sezione 3.3, è stata condotta un'analisi quantitativa per misurare la riduzione del rischio di propagazione (*Ripple Effect*) tra i moduli del sistema. Le metriche selezionate derivano dai principi di *Software Package Metrics* applicati ai componenti architetturali.

3.5.1 Definizione delle Metriche

Per quantificare l'impatto, si utilizzano i seguenti indicatori:

- **Cross-Module Coupling (XMC)**: Numero totale di dipendenze dirette che attraversano i confini dei moduli. Un valore elevato indica un sistema "spaghetti code" dove la modifica di un modulo impatta gli altri.
- **Afferent Coupling Esterne (Ca_{ext})**: Numero di componenti esterni al modulo che dipendono da esso. Misura la responsabilità e la criticità del modulo verso l'esterno.
- **Ripple Scope Index (RSI)**: Grado stimato del numero di file da modificare in caso di refactoring dell'interfaccia del modulo Utenti.

3.5.2 Analisi del coupling

La Tabella 3.5 riporta i valori metrici della Baseline. Il calcolo è stato effettuato contando gli elementi non nulli ($A_{ij} \neq 0$) che collegano righe e colonne appartenenti a moduli diversi.

Tabella 3.5: Confronto Metriche di Accoppiamento Inter-Modulo

Metrica	**Baseline**
Cross-Module Coupling (XMC)	8
Ca_{ext} **(Utenti.Model)**	5
Ripple Scope Index (RSI)	Alto

Interpretazione dei risultati

I valori riportati nella Tabella 3.5 consentono di delineare in modo chiaro il profilo di accoppiamento inter-modulo della Baseline. Il valore di *Cross-Module Coupling* (XMC), pari a 5, indica la presenza di cinque connessioni strutturali tra moduli distinti. Tale numero, pur non elevato in termini assoluti, risulta significativo se rapportato alla dimensione del sistema e alla natura dei moduli coinvolti: esso evidenzia che una parte non

trascurabile delle funzionalità dipende da componenti esterni al proprio dominio logico, con potenziali ripercussioni sulla manutenibilità e sulla propagazione degli impatti.

L'analisi dei valori di Ca_{ext} permette di qualificare ulteriormente la natura di tali dipendenze. Il modulo *Utenti.Model* presenta un valore pari a 4, segno che le sue entità vengono frequentemente referenziate da altri moduli. Ciò suggerisce un ruolo centrale del modello utenti nella Baseline, che funge da punto di intersezione semantica per più funzionalità applicative. Sebbene tale centralità sia coerente con la natura del dominio, essa comporta un accoppiamento strutturale che può amplificare gli effetti di modifiche future, aumentando il rischio di regressioni.

Il valore di Ca_{ext} relativo al livello *Utenti.Service/Facade*, pari a 2, indica invece un accoppiamento più contenuto verso le interfacce di servizio. Questo dato suggerisce che, nella Baseline, i servizi utenti non sono ancora pienamente utilizzati come punto di accesso controllato al dominio, lasciando che altri moduli interagiscano direttamente con il modello. Tale configurazione riduce l'incapsulamento e limita la possibilità di introdurre logiche trasversali (validazioni, controlli di sicurezza, caching) senza propagare modifiche ai moduli dipendenti.

Infine, il *Ripple Scope Index* (RSI), pari a 5 file, quantifica l'ampiezza potenziale della propagazione degli impatti: una modifica localizzata in uno dei punti di accoppiamento potrebbe richiedere interventi in cinque file distribuiti tra moduli diversi. Questo valore conferma che la Baseline presenta un accoppiamento non trascurabile, in grado di influenzare la stabilità complessiva del sistema e di aumentare il costo evolutivo.

Nel complesso, le metriche evidenziano una struttura in cui il modulo utenti, in particolare il suo modello, costituisce un nodo centrale di dipendenze. Tale configurazione, pur funzionale, risulta meno allineata ai principi di modularità e separazione delle responsabilità, motivando la necessità di un refactoring volto a ridurre l'accoppiamento e a migliorare la resilienza architetturale del sistema.

3.6 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. A ciò è stato applicato un processo di filtro verso componenti non direttamente sviluppate dal team (come eccezioni o librerie) e sono state invece aggiunte componenti logicamente correlate alla manutenzione, come il DatabasePopulator e il percistence.xml.

3.6.1 Backward Dependencies Analysis of impacted components

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	–
DocenteService	jdk	–
CoordinatoreService	jsp	Account.jsp.
CoordinatoreService	test	CoordinatoreServiceTest, CoordinatoreServiceBenchmark.
CoordinatoreService	service	–

Componente	Package	Backward Dependencies
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 3.6: Backward dependencies per componente e package

3.6.2 Forward Dependencies Analysis of impacted components

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

Componente	Package	Forward Dependencies
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.
CoordinatoreService	jdk	InitialContext, NamingException, List, String, Object.

Componente	Package	Forward Dependencies
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 3.7: Forward dependencies per componente e package

3.6.3 CIS dei file di codice sorgente

Il seguente CIS, come detto in precedenza, è stato stilato coinvolgendo all'interno del set tutte le componenti rinvenute durante l'analisi delle dipendenze in avanti e all'indietro.

Elemento citato	Origine (BD/FD)
Account.jsp	BD StudenteService, BD DocenteService, BD CoordinatoreService, BD PersonaleTAService
Accademico	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD UtenteService
AccademicoDAO	FD UtenteService
AccademicoRemote	FD AccademicoService

Elemento citato	**Origine (BD/FD)**
AlreadyExistentUserException	FD StudenteService, FD DocenteService, FD CoordinatoreService
AttivaUtentiServlet	BD AccademicoService
AuthenticationException	FD UtenteService
chatServlet	BD AccademicoService
ConversazioniServlet	BD AccademicoService
Coordinatore	FD CoordinatoreService
CoordinatoreRemote	FD CoordinatoreService
CoordinatoreServiceBenchmark	BD CoordinatoreService
CoordinatoreServiceTest	BD CoordinatoreService
CorsoLaurea	FD StudenteService
Docente	FD DocenteService
DocenteRemote	FD DocenteService
DocenteServiceBenchmark	BD DocenteService
DocenteServiceTest	BD DocenteService
Exception	FD PersonaleTAService
GetAttivati	BD AccademicoService
GetEmailServlet	BD AccademicoService
GetNonAttivati	BD AccademicoService
IncorrectUserSpecification	FD StudenteService, FD DocenteService, FD CoordinatoreService
InitialContext	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService
invioMessaggioServlet	BD AccademicoService
JMH benchmark	BD UtenteService
List	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService
LoginServlet	BD AccademicoService, BD PersonaleTAService

Elemento citato	**Origine (BD/FD)**
NamingException	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService
NoResultException	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService, FD UtenteService
NotFoundUserException	FD StudenteService, FD DocenteService, FD CoordinatoreService
Object	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService
PersonaleTA	FD PersonaleTAService, FD UtenteService
PersonaleTARemote	FD PersonaleTAService
PrintStream	FD AccademicoService
Stateless	FD AccademicoService, FD PersonaleTAService, FD UtenteService
String	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD PersonaleTAService, FD UtenteService
Studente	FD StudenteService
StudenteRemote	FD StudenteService
StudenteServiceTest	BD StudenteService
Utente	FD AccademicoService, FD StudenteService, FD DocenteService, FD CoordinatoreService, FD UtenteService
UtenteService	BD PersonaleTAService

Elemento citato	Origine (BD/FD)
UtenteServiceTest	BD AccademicoService, BD UtenteService
vari test controller	BD AccademicoService

Tabella 3.8: Candidate Impact Set

3.6.4 Pulizia del CIS

Durante l'analisi delle backward e forward dependencies sono emersi anche elementi che non rappresentano artifact software del progetto UniClass, bensì eccezioni di libreria o tipi infrastrutturali del JDK/EJB. Tali elementi, pur comparendo nei grafi di dipendenza, **non vengono inclusi nel CIS effettivo** utilizzato per le misurazioni quantitative.

Eccezioni escluse

- AlreadyExistentUserException
- AuthenticationException
- IncorrectUserSpecification
- NoResultException
- NotFoundUserException
- Exception

Tipi infrastrutturali esclusi

- String
- Object
- List
- Stateless
- PrintStream
- InitialContext
- NamingException

Questi elementi non rappresentano componenti del sistema, pertanto non vengono considerati nel calcolo delle metriche di Impact Analysis.

3.6.5 CIS completo

Il seguente set rappresenta il Candidate Impact Set (CIS) depurato dagli elementi infrastrutturali (JDK, EJB) e dalle eccezioni di libreria, focalizzandosi esclusivamente sugli artifact software del sistema UniClass[cite: 874, 892].

Artifact nel CIS (Ripulito + SIS + Integrazioni Complete)
Account.jsp
Accademico
AccademicoDAO
AccademicoRemote
AttivaUtentiServlet
chatServlet
ConversazioniServlet
AccademicoServiceTest
StudenteServiceTest
DocenteServiceTest
CoordinatoreServiceTest
UtenteServiceTest
vari test controller
StudenteService
DocenteService
CoordinatoreService
UtenteService
DocenteServiceBenchmark
CoordinatoreServiceBenchmark
PersonaleTAService
LoginServlet
AccademicoService
Studente
Docente
Coordinatore
PersonaleTA
StudenteRemote
DocenteRemote
CoordinatoreRemote

Artifact nel CIS (Ripulito + SIS + Integrazioni Complete)
PersonaleTARemote
Utente
CorsoLaurea
InvioMessaggioServlet
Conversazioni.jsp
LogoutServlet
GetAttivati
GetNonAttivati
CercaOrario
DatabasePopulator
UtenteDAO
DocenteDAO
StudenteDAO
CoordinatoreDAO
PersonaleTADAO
GetEmailServlet
Persistence.xml
Modulo Conversazioni (Model, Service, Test)
Messaggio.java
MessaggioTest.java
MessaggioService.java
MessaggioServiceTest.java
MessaggioServiceBenchmark.java
Topic.java
TopicService.java
TopicServiceTest.java
TopicServiceBenchmark.java
ConversazioniServletTest.java
chatServeltTest.java
invioMessaggioServletTest.java
Modulo Orari (Model, Service, Test)
Corso.java
Lezione.java

Artifact nel CIS (Ripulito + SIS + Integrazioni Complete)
Resto.java
AnnoDidatticoService.java
AulaService.java
CorsoLaureaService.java
CorsoService.java
LezioneService.java
RestoService.java
AulaServiceTest.java
CorsoLaureaServiceTest.java
CorsoServiceTest.java
LezioneServiceTest.java
RestoServiceTest.java
EdificioServletTest.java
cercaOrarioTest.java
getAnnoTest.java
getRestoTest.java
BenchmarkAnnoDidatticoService.java
BenchmarkAulaService.java
BenchmarkCorsoLaureaService.java
BenchmarkCorsoService.java
BenchmarkLezioneService.java
BenchmarkRestoService.java
Benchmark e Mocks (Utenti)
PersonaleTAServiceBenchmark.java
UtenteServiceBenchmark.java
MockAccademicoDAO.java
MockCoordinatoreDAO.java
MockDocenteDAO.java
MockPersonaleTADAO.java

Tabella 3.9: Candidate Impact Set finale (Software Artifacts)

La cardinalità totale del CIS è 88.

3.6.6 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.

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.

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
Modello dati Utente/Accademico	Ridefinizione delle relazioni che influenza le chiamate DAO e Service.	P1, P4

Tabella 3.10: Primary Impact Set basato su forward e backward dependencies

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 3.11: Secondary Impact Set basato su relazioni di chiamata

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;
- analisi delle **dependencies matrix** con IntelliJ;
- calcolo delle **reachibility matrix** tramite algoritmo di **Warshall**
- analisi del **ripple effect**
- 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.

3.7 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.
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.

Componente	**Package**	**Dipendenze verso AccademicoService**
Conversazioni.jsp, chat.jsp	jsp	Richiedono dati ottenuti tramite AccademicoService.

Tabella 3.12: CIS suddiviso per package per le dipendenze verso `AccademicoService`

Capitolo 4

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.

4.0.1 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.

4.0.2 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 *Facade* (`UserDir`) per centralizzare azioni comuni ai servizi. 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).

4.1 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.

4.1.1 Mapping CIS – Strategia di Test

La Tabella 4.1 riassume la relazione tra i componenti del CIS e le attività di testing previste, evidenziandone la priorità.

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

Tabella 4.1: Mapping tra CIS e strategia di regression testing

Capitolo 5

Actual Impact Set

5.1 Metodologia di Estrazione dell'Actual Impact Set (AIS)

L'Actual Impact Set (AIS) è stato ricostruito mediante un'analisi differenziale tra la versione *baseline* del progetto UniClass e la versione *post-refactoring* contenente l'implementazione della Change Request CR-UniClass. Poiché il refactoring architetturale del modulo Utenti è stato eseguito nella fork locale l'AIS è stato estratto utilizzando la funzionalità di *Comparing changes* messa a disposizione dall'interfaccia web di GitHub.

In particolare, sono stati selezionati due commit significativi nella repository originale:

- **Commit pre-refactoring**: ultimo commit antecedente alla rimozione della gerarchia `Utente` e dei servizi specializzati;
- **Commit post-refactoring**: commit contenente l'introduzione di `UserDirectory`, la ristrutturazione del modello dati e la modifica dei servizi e DAO del modulo Utenti.

Attraverso la vista di confronto (*Compare*) di GitHub, è stato generato l'elenco completo dei file modificati tra tali versioni. Tale lista rappresenta l'insieme degli artifact effettivamente impattati dal refactoring e costituisce, a tutti gli effetti, l'Actual Impact Set.

L'output risultante include:

- classi del modello Utenti (`Utente`, `Accademico`, `Studente`, `Docente`, `Coordinatore`, `PersonaleTA`);
- servizi e DAO del modulo Utenti;
- servlet e controller coinvolti nei flussi UC1 e UC9;
- JSP correlate alla gestione dell'utenza;

- file di configurazione quali `persistence.xml` e `resources.xml`;
- test unitari, di integrazione e benchmark JMH/JMeter;
- componenti dei moduli *Conversazioni* e *Orari* impattati per propagazione.

L'AIS così ottenuto è stato successivamente confrontato con il Candidate Impact Set (CIS) per valutare la copertura dell'analisi statica e la correttezza della previsione dell'impatto. Tale confronto ha permesso di identificare eventuali falsi positivi e falsi negativi, fornendo una misura quantitativa dell'efficacia del processo di Impact Analysis.

5.2 Actual Impact Set and Propagation analysis

5.2.1 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.

5.2.2 Actual Impact Set

La cardinalità totale dell'AIS è 185.

Artifact	Change Type	Reason for Change	Original Issue	Impact
Documentazione				
`ImpactAnalysis_UniClass_v1.0.pdf`	Add	Generazione AIS	Mancanza report AIS	Disponibilità report AIS completo
`TCS_v1_UniClass.pdf`	Add	Test Case Specification	Mancanza TCS	Disponibilità TCS per moduli
`TestPlan_UniClass.pdf`	Add	Test Plan	Mancanza test plan	Piano di test documentato
`ObjectDesignDocument_UniClass.pdf`	Move	Allineamento versioni	Versione precedente obsoleta	Centralizzazione ODD
`PresentazioneUniClass.pdf`	Move	Versioning	Versione precedente obsoleta	Disponibilità presentazione aggiornata
`ProblemStatement_UniClass.pdf`	Move	Versioning	Versione precedente obsoleta	Problem statement centralizzato
`RequirementAnalysisDocument_UniClass.pdf`	Move	Versioning	Versione precedente obsoleta	Requisiti centralizzati
`SystemDesignDocument_UniClass.pdf`	Move	Versioning	Versione precedente obsoleta	Design centralizzato

Artifact	Change Type	Reason for Change	Original Issue	Impact
TestExcecutionReport_UniClass.pdf	Move	Versioning	Report test legacy	Disponibilità report test eseguiti
TestPlanAndSpecificationCases_UniClass.pdf	Move	Versioning	Versione precedente obsoleta	Test plan e TCS centralizzati
Benchmark / Baseline				
summary_auth.csv	Add	Performance testing	Nessun CSV baseline	Analisi load auth
summary_load.csv	Add	Performance testing	Nessun CSV baseline	Analisi load generale
summary_spike.csv	Add	Performance testing	Nessun CSV baseline	Analisi spike
summary_stress.csv	Add	Performance testing	Nessun CSV baseline	Analisi stress
jmh-result-anno-didattico.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Anno-Didattico
jmh-result-aula.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Aula
jmh-result-coordinatore.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Coordinatore
jmh-result-corso-laurea.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Corso-Laurea
jmh-result-docente.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Docente

Artifact	Change Type	Reason for Change	Original Issue	Impact
jmh-result-lezione.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Lezione
jmh-result-messaggio.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Messaggio
jmh-result-personale-ta.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH PersonaleTA
jmh-result-resto.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Resto
jmh-result-topic.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Topic
jmh-result-utente.json	Add	Benchmark JMH	Nessun baseline JMH	Baseline JMH Utente
baseline_dependency.csv	Add	Dependency analysis	Nessun report baseline	Analisi dipendenze CSV
baseline_dependency.xml	Add	Dependency analysis	Nessun report baseline	Analisi dipendenze XML
dependency.xml	Add	Dependency analysis	Nessun report baseline	Analisi dipendenze XML aggiornato
Moduli				
ConversazioniServlet.java	Modify	Correzione logica LazyInitialization	LazyInitializationException	Stabilità modulo Conversazioni
chatServlet.java	Modify	Correzione request/session e preload relazioni LAZY	NPE e LazyInitializationException	Chat funzionante

Artifact	**Change Type**	**Reason for Change**	**Original Issue**	**Impact**
`invioMessaggioServlet.java`	Modify	Allineamento nuovo modello utenti	Dipendenza servizi rimossi	Coerenza con UserDirectory
`Messaggio.java`	Modify	Annotazioni ORM	Errore SQL tabella non trovata	Corretta mappatura ORM
`MessaggioService.java`	Modify	Refactoring	Dipendenze obsolete	Servizio coerente
`TopicService.java`	Modify	Refactoring	Dipendenze obsolete	Servizio coerente
`AulaServlet.java`	Modify	Centralizzazione caricamento aule	LazyInitializationException	Accesso sicuro a dati
`EdificioServlet.java`	Replace	Separazione logica/presentazione	NPE, LazyInitializationException	UI stabile
`cercaOrario.java`	Modify	Correzione chiamate AJAX	AJAX falliva	Caricamento corretto
`getAnno.java`	Create	Nuovo endpoint	Mancanza controller dedicato	Routing corretto
`getResto.java`	Create	Nuovo endpoint	Mancanza controller dedicato	Popolamento dati asincrono corretto
`Corso.java`	Modify	Refactoring modello	Dipendenze obsolete	Modellazione coerente
`CorsoLaurea.java`	Modify	Refactoring modello	Dipendenze obsolete	Modellazione coerente
`Lezione.java`	Modify	Refactoring modello	Dipendenze obsolete	Modellazione coerente

Artifact	**Change Type**	**Reason for Change**	**Original Issue**	**Impact**
`Resto.java`	Modify	Refactoring modello	Dipendenze obsolete	Modellazione coerente
`AnnoDidatticoService.java`	Modify	Allineamento nuovo modello utenti	Dipendenze obsolete	Servizio coerente
`AulaService.java`	Modify	Refactoring logica accesso	Accesso incoerente ai DAO	Servizio stabile
`CorsoLaureaService.java`	Modify	Allineamento nuovo modello utenti	Dipendenze obsolete	Servizio coerente
`CorsoService.java`	Modify	Refactoring logica accesso	Accesso incoerente ai DAO	Servizio stabile
`LezioneService.java`	Modify	Refactoring logica accesso	Accesso incoerente ai DAO	Servizio stabile
`RestoService.java`	Modify	Refactoring logica accesso	Accesso incoerente ai DAO	Servizio stabile
Test				
`ConversazioniServletTest.java`	Modify	Aggiornamento test per nuove dipendenze	Test legacy fallivano	Test coerente
`chatServeltTest.java`	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
`invioMessaggioServletTest.java`	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
`MessaggioTest.java`	Modify	Aggiornamento test	Test legacy fallivano	Test coerente

Artifact	Change Type	Reason for Change	Original Issue	Impact
MessaggioServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
TopicServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
EdificioServletTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
cercaOrarioTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
getAnnoTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
getRestoTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
AulaServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
CorsoLaureaServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
CorsoServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
LezioneServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
RestoServiceTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
AttivaUtentiServletTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
GetAttivatiTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
GetEmailServletTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
GetNonAttivatiTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
LoginServletTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
AccademicoTest.java	Modify	Aggiornamento test	Test legacy fallivano	Test coerente

Artifact	**Change Type**	**Reason for Change**	**Original Issue**	**Impact**
`CoordinatoreTest.java`	Remove	Modello legacy eliminato	DAO/Service rimossi	Pulizia test
`DocenteTest.java`	Remove	Modello legacy eliminato	DAO/Service rimossi	Pulizia test
`PersonaleTATest.java`	Remove	Modello legacy eliminato	DAO/Service rimossi	Pulizia test
`StudenteTest.java`	Remove	Modello legacy eliminato	DAO/Service rimossi	Pulizia test
`AccademicoServiceTest.java`	Remove	Service legacy eliminato	Dipendenze obsolete	Pulizia test
`CoordinatoreServiceTest.java`	Remove	Service legacy eliminato	Dipendenze obsolete	Pulizia test
`DocenteServiceTest.java`	Remove	Service legacy eliminato	Dipendenze obsolete	Pulizia test
`PersonaleTAServiceTest.java`	Remove	Service legacy eliminato	Dipendenze obsolete	Pulizia test
`StudenteServiceTest.java`	Remove	Service legacy eliminato	Dipendenze obsolete	Pulizia test
`UserDirectoryTest.java`	Add	Nuovo test per UserDirectory	Nessun test baseline	Verifica correttezza UserDirectory
`UtenteServiceTest.java`	Modify	Aggiornamento test	Test legacy fallivano	Test coerente
`AccademicoDAOTest.java`	Modify	Aggiornamento DAO test	Dipendenze obsolete	Test coerente
`CoordinatoreDAOTest.java`	Remove	DAO legacy eliminato	DAO rimossi	Pulizia test
`DocenteDAOTest.java`	Remove	DAO legacy eliminato	DAO rimossi	Pulizia test
`PersonaleTADAOTest.java`	Remove	DAO legacy eliminato	DAO rimossi	Pulizia test
`StudenteDAOTest.java`	Remove	DAO legacy eliminato	DAO rimossi	Pulizia test

Artifact	Change Type	Reason for Change	Original Issue	Impact
UtenteDAOTest.java	Modify	Aggiornamento DAO test	Dipendenze obsolete	Test coerente
Benchmark Tests JMH/JMeter				
uniclass-auth-spike-test.jmx	Add	Performance test JMeter	Nessun baseline	Spike auth test
uniclass-load-test.jmx	Add	Performance test JMeter	Nessun baseline	Load test generale
uniclass-spike-test.jmx	Add	Performance test JMeter	Nessun baseline	Spike test generale
uniclass-stress-test.jmx	Add	Performance test JMeter	Nessun baseline	Stress test generale
benchmarkMessaggioService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni MessaggioService
benchmarkTopicService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni TopicService
BenchmarkAnnoDidatticoService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni AnnoDidatticoService
BenchmarkAulaService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni AulaService
BenchmarkCorsoLaureaService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni CorsoLaureaService
BenchmarkCorsoService.java	Add	JMH benchmark	Nessun baseline	Misura prestazioni CorsoService

Artifact	Change Type	Reason for Change	Original Issue	Impact
`BenchmarkLezioneService.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni LezioneService
`BenchmarkRestoService.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni RestoService
`CoordinatoreServiceBenchmark.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni CoordinatoreService
`DocenteServiceBenchmark.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni DocenteService
`PersonaleTAServiceBenchmark.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni PersonaleTAService
`UtenteServiceBenchmark.java`	Add	JMH benchmark	Nessun baseline	Misura prestazioni UtenteService
`MockAccademicoDAO.java`	Add	DAO mock	Necessità test isolati	Mock DAO per test unitari
`MockCoordinatoreDAO.java`	Add	DAO mock	Necessità test isolati	Mock DAO per test unitari
`MockDocenteDAO.java`	Add	DAO mock	Necessità test isolati	Mock DAO per test unitari
`MockPersonaleTADAO.java`	Add	DAO mock	Necessità test isolati	Mock DAO per test unitari

5.2.3 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.

Capitolo 6

Valutazione Quantitativa dell'Impact Analysis

In conformità con quanto descritto nel manuale di riferimento e con la struttura del presente report, la valutazione dell'accuratezza dell'Impact Analysis viene completata introducendo metriche quantitative standard: *Precision* e *Recall*. Tali metriche consentono di misurare in modo oggettivo la qualità dell'identificazione degli elementi impattati, sia a livello di *Class Impact Set (CIS)* sia di *Architectural Impact Set (AIS)*, come riportato rispettivamente nelle Tabelle **??**:contentReference[oaicite:0]index=0 e **??**:contentReference[oaicite:1]index=1.

Dati di Partenza

Dall'analisi quantitativa risultano i seguenti valori:

- $|AIS| = 180$
- $|CIS| = 88$
- Componenti presenti nel CIS ma non nell'AIS: $\emptyset$ (Il CIS è un sottoinsieme dell'AIS).
- Componenti presenti nell'AIS ma non nel CIS:
  - **Documentazione e Deliverable**: File PDF di analisi, design e reportistica (`ImpactAnalysis_UniClass_v1.0.pdf`, `TCS_v1_UniClass.pdf`, ecc.).
  - **Baseline e Dati di Supporto**: Risultati benchmark (`.csv`, `.json`) e report dipendenze (`dependency.xml`).
  - **Script di Performance Testing**: File di configurazione JMeter (`uniclass-load-test.jmx` ecc.).
  - **Controller Modulo Orari**: Servlet identificate come impattate nell'AIS ma non classificate nel CIS (`AulaServlet.java`, `EdificioServlet.java`, `getAnno.java`, `getResto.java`).

Ne consegue che:

$$|CIS \cap AIS| = 88$$

Definizione delle Metriche

Si adottano le metriche classiche di *precision* e *recall*:

$$Precision = \frac{|CIS \cap AIS|}{|CIS|}$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|}$$

Calcolo delle Metriche

Precision

$$Precision = \frac{88}{88} = 1.00$$

Il valore di precision pari a 1 indica che tutte le componenti identificate nel CIS sono risultate effettivamente impattate. Non si registrano *false positive*. L'analisi ha dunque evitato sovrastime dell'impatto.

Recall

$$Recall = \frac{88}{180} \approx 0.47$$

Il valore di recall pari a circa 0.47 evidenzia la presenza di *false negative*, ovvero componenti effettivamente impattate ma non incluse nel CIS iniziale. Questo risultato va interpretato alla luce del fatto che molti file dell'AIS sono risultati o report. Il vero fattore che ha influenzato la recall riguarda il modulo Orari e la mancata presa in considerazione dell'effetto domino dovuto alle modifiche su Orari.model.

L'Impact Analysis condotta presenta le seguenti caratteristiche:

- **Alta precisione (100%)**: il metodo basato su forward e backward dependencies ha consentito di individuare esclusivamente componenti realmente coinvolte.
- **Recall moderato (47%)**: una parte dell'impatto si è propagata verso moduli non strettamente connessi al dominio Utenti (es. gestione orario, aule, conversazioni, JSP e script client-side), non completamente catturati dall'analisi statica iniziale.

In termini qualitativi, l'approccio adottato si dimostra:

- *Conservativo rispetto ai falsi positivi*;
- *Parzialmente limitato nella cattura delle propagazioni trasversali*, in particolare verso componenti di presentazione (JSP, JS) e servizi correlati indirettamente.

Il risultato complessivo suggerisce che l'approccio basato sui grafi di dipendenza fornisce un'ottima base per l'identificazione del *core impact*, ma dovrebbe essere integrato con:

- analisi dinamica dei flussi runtime,
- verifica delle dipendenze lato presentazione,
- analisi delle dipendenze indirette tra moduli funzionali.

6.1 Backward Dependency Graphs

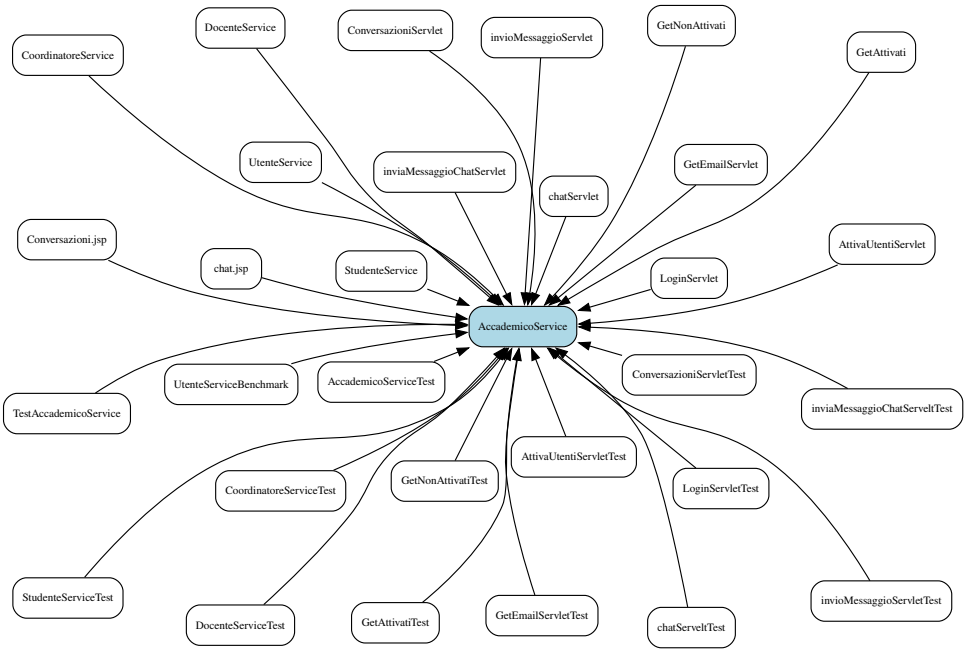


Figura 6.1: Backward Dependencies del God Service

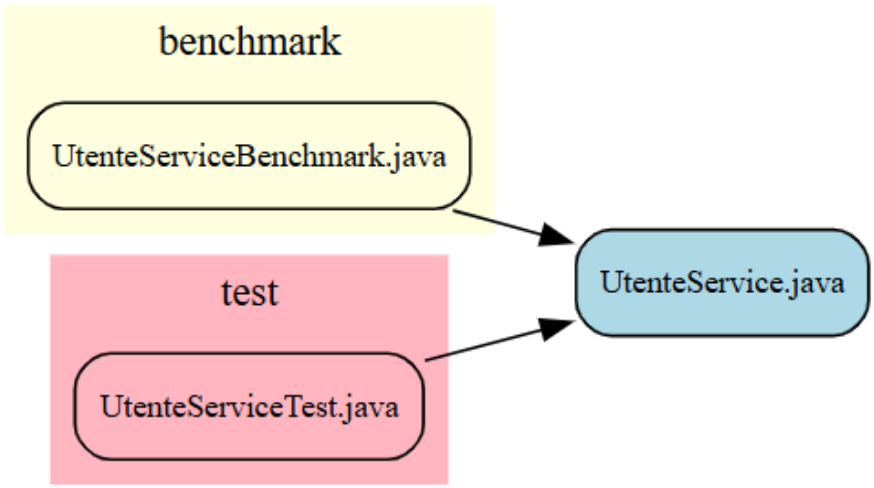


Figura 6.2: UtenteService backward dependencies

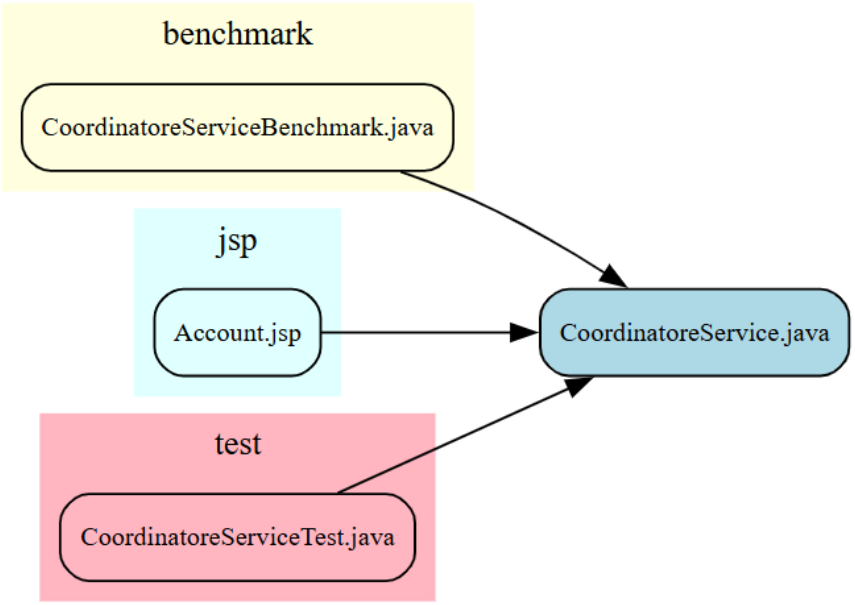
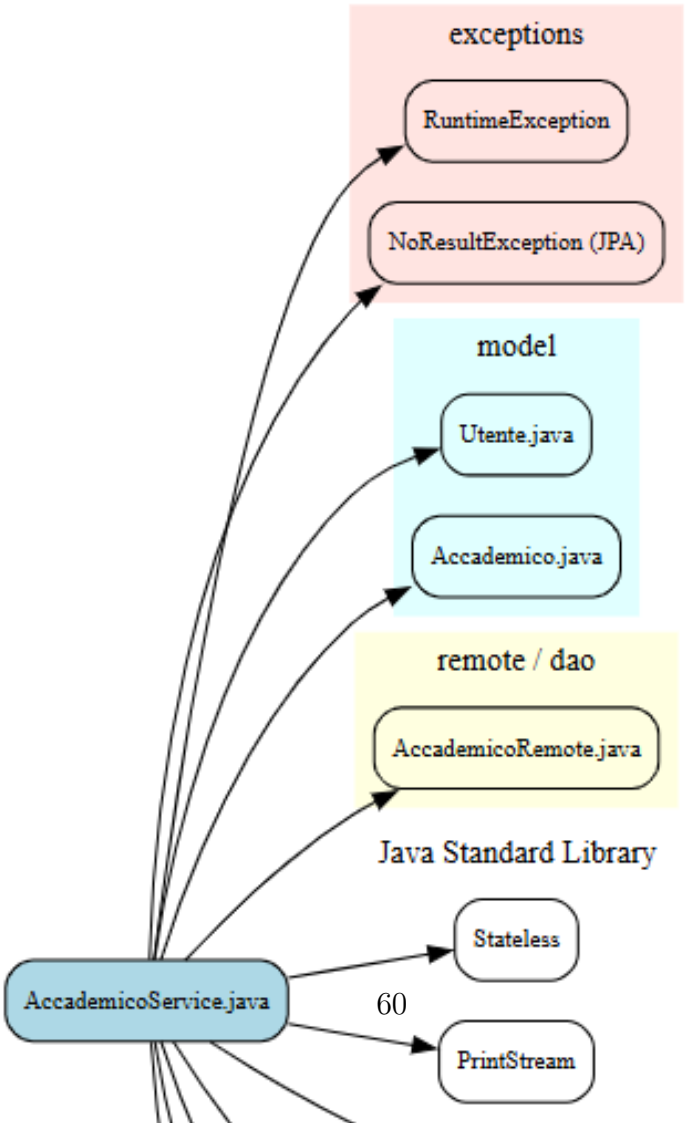


Figura 6.3: Backward Dependencies della specializzazione Coordinatore Service

6.2 Forward Dependency Graphs



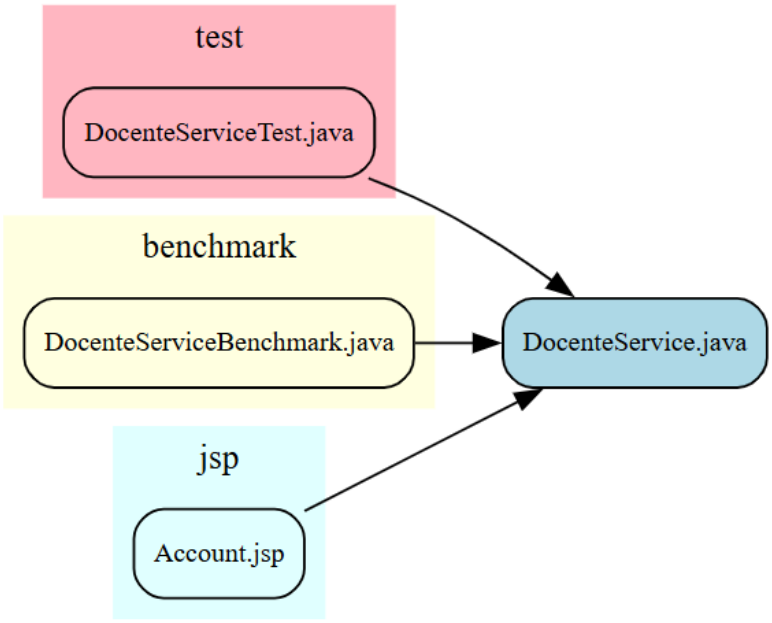
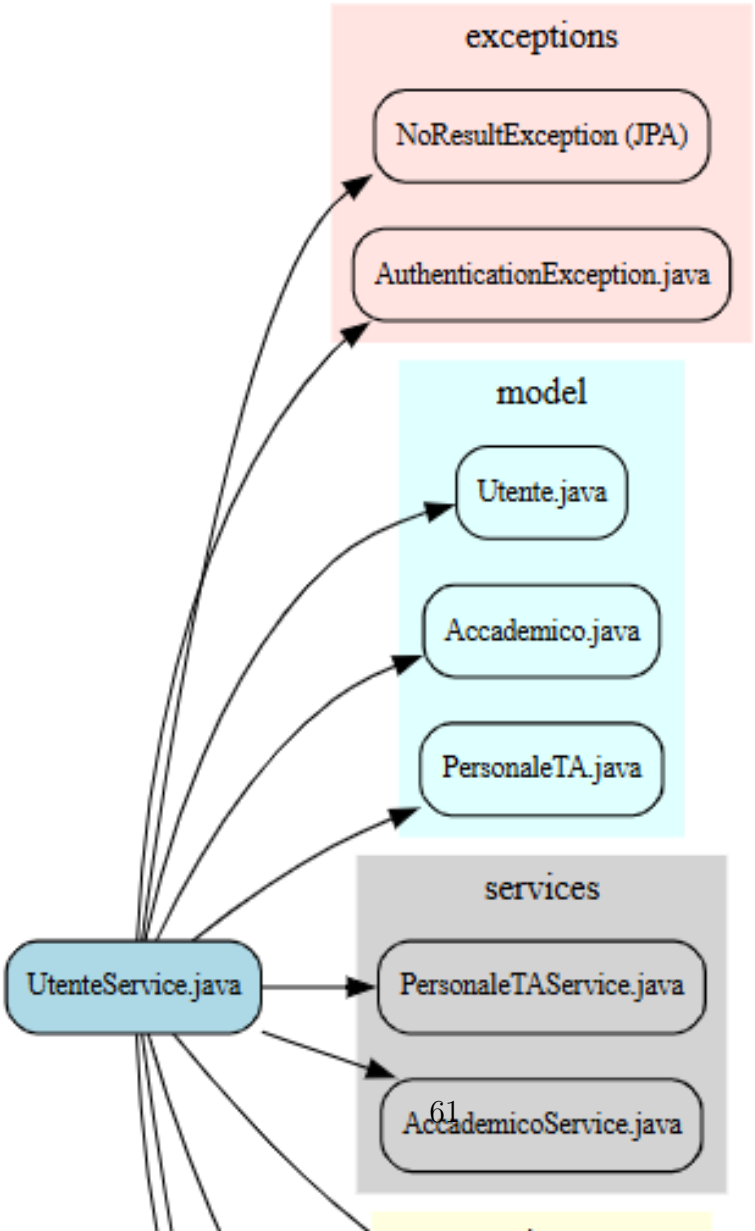


Figura 6.4: Backward dependencies della specializzazione Docente Service



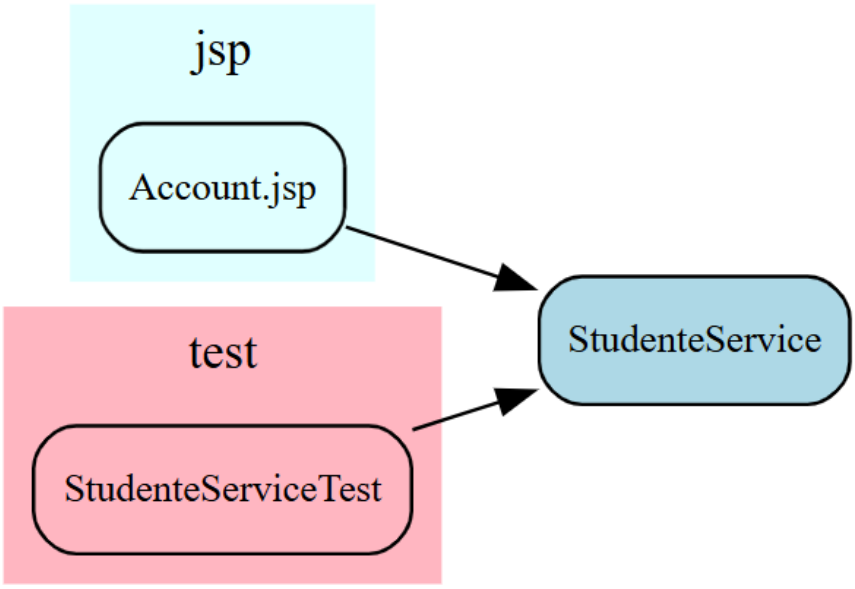
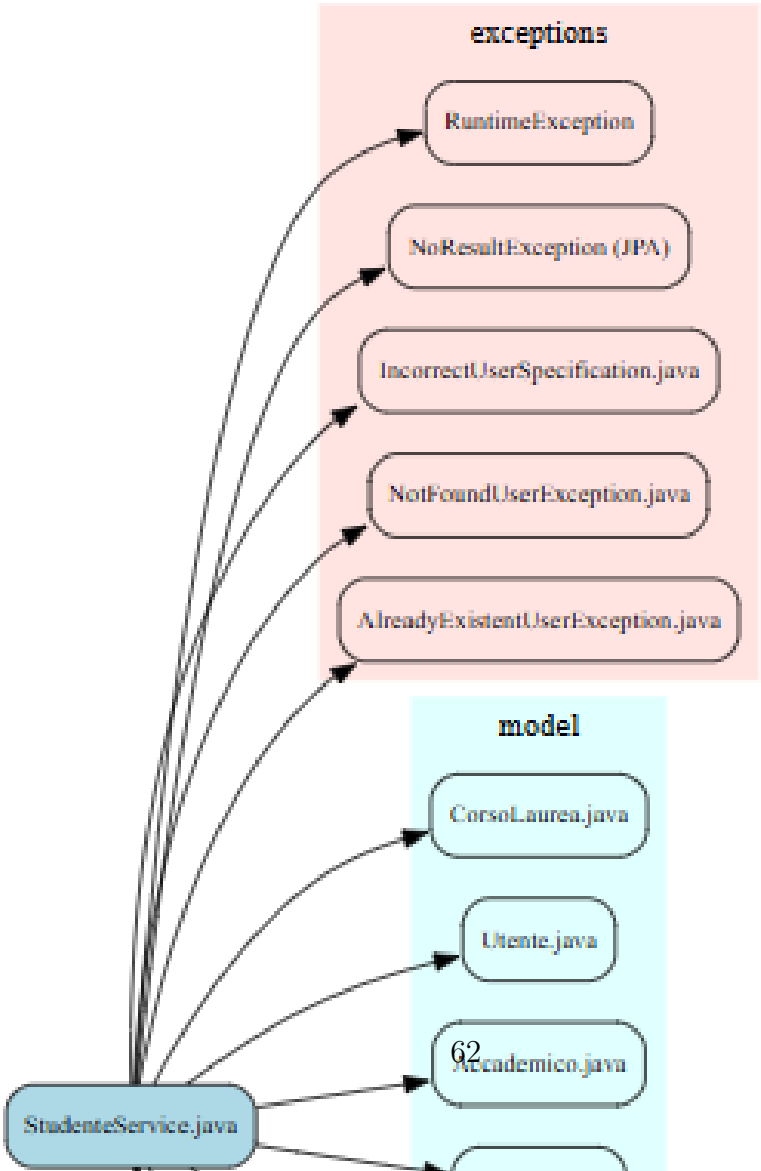


Figura 6.5: Backward Dependencies della specializzazione Studente Service



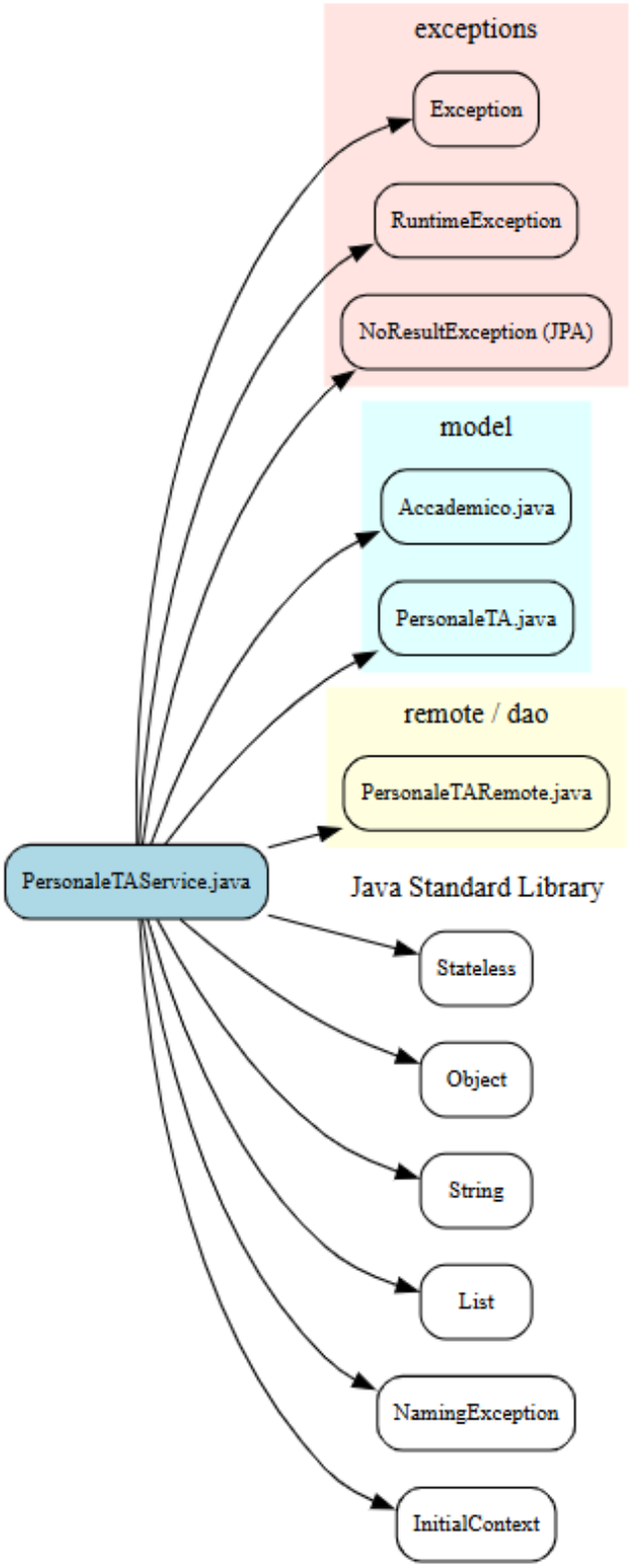


Figura 6.10: Forward dependency graph di `PersonaleTAService`

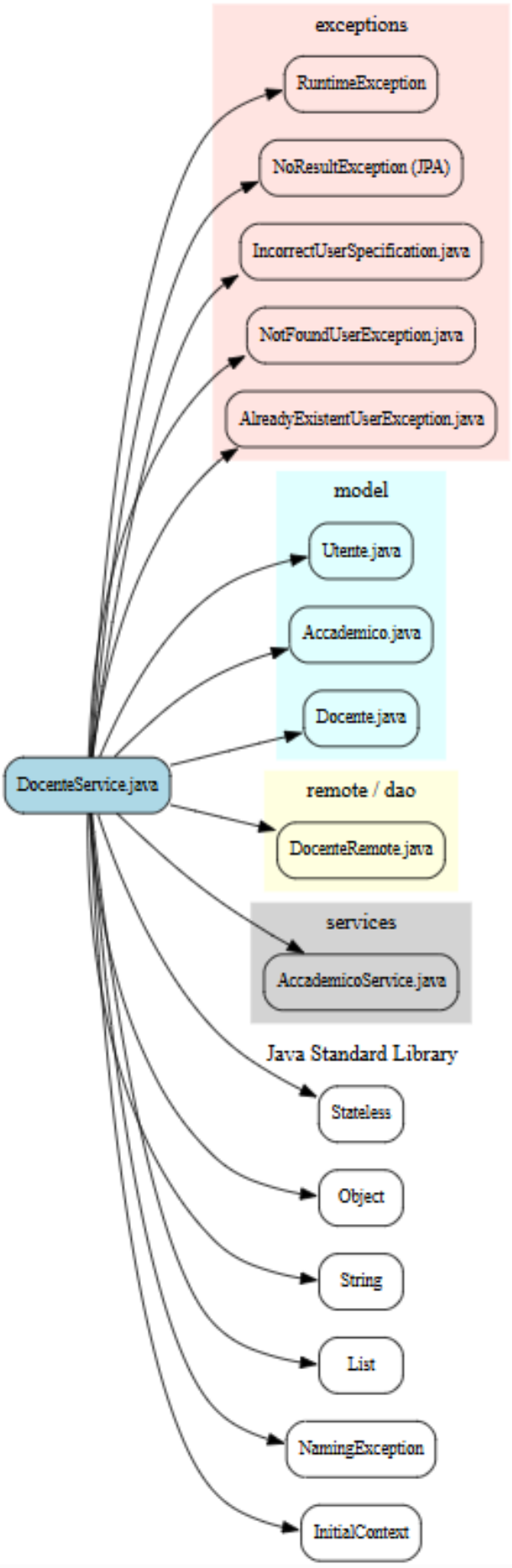


Figura 6.11: Forward dependency graph di `DocenteService`

6.3 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.

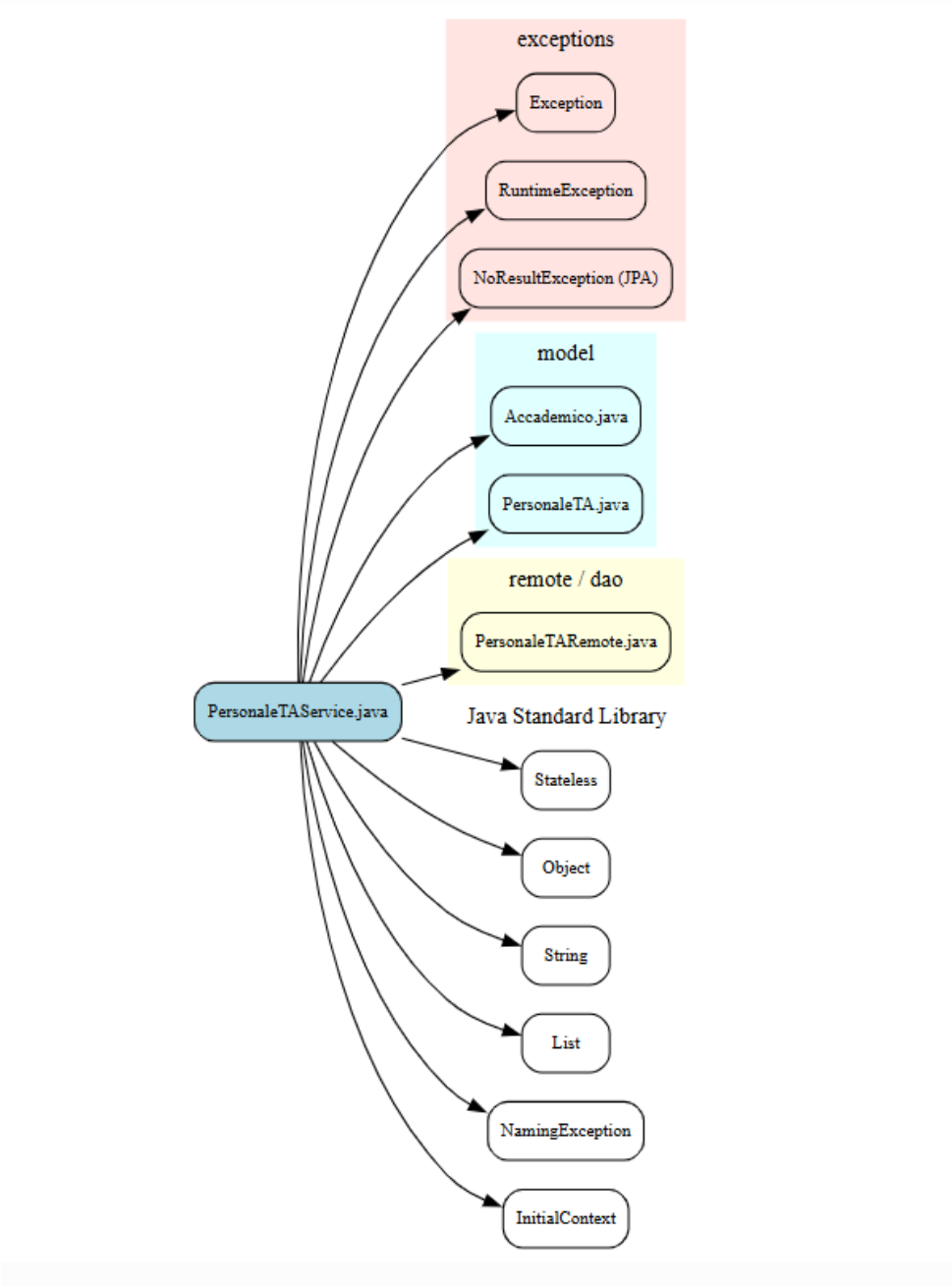


Figura 6.6: Backward Dependencies di PersonaleTA Service