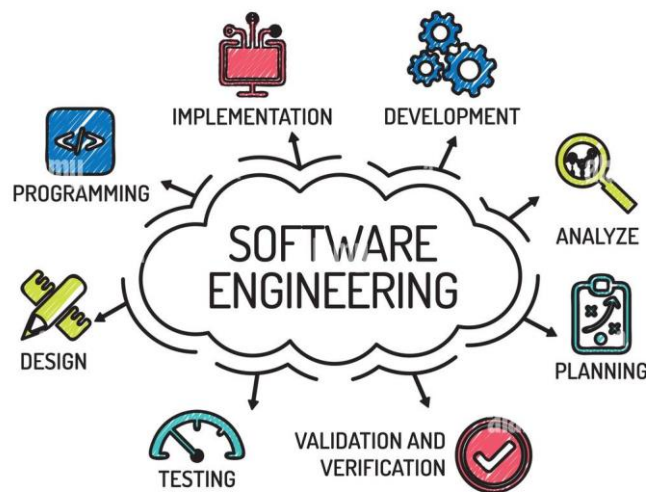


Università degli Studi di Palermo

Ingegneria del Software (9 CFU)

# System Design Document (SDD)

Progettazione di un software per il supporto ad un'azienda di servizi al cittadino



Un progetto realizzato da:

**Bova Gabriele**  
**Carini Riccardo**  
**Cimino Giuseppe**  
**La Rosa Mazza Nicolò**

**CdL: Ingegneria Informatica LT**  
**Anno Accademico: 2022/2023**

# Sommario

1. Introduction	3
1.1 Purpose of the system	3
1.2 Design goals	3
1.3 Definitions, acronyms, and abbreviations	3
1.4 References	4
1.5 Overview	4
2. Current software architecture	4
3. Proposed software architecture	5
3.1 Overview	5
3.2 Subsystem decomposition	5
3.3 Hardware/software mapping	24
3.4 Persistent data management	26
3.4.1 Progettazione Modello E-R	26
3.4.2 Traduzione Modello E-R	28
3.5 Access control and security	36
3.6 Global software control	37
3.7 Boundary conditions	37
4. Subsystem services	37
5. Glossary	37

## 1. Introduction

Il System Design Document (SDD) è un documento che descrive come i requisiti funzionali e non funzionali identificati in fase di analisi dei requisiti si trasformano in specifiche tecniche di progettazione del sistema, che verranno usate per progettare il sistema stesso. Da una prima panoramica ad alto livello sull'architettura scelta, passeremo a descrivere e ad analizzare più a basso livello i sottosistemi nei quali decomporremo il sistema e approfondiremo inoltre come avviene la gestione dei dati persistenti. Ci teniamo a ricordare che l'analisi di questi punti non procederà in maniera algoritmica: come spesso accade nelle discipline ingegneristiche è necessario fare delle scelte e dei compromessi, ad esempio favorendo determinati requisiti piuttosto che altri. Inoltre non sempre è facile prevedere con largo anticipo ostacoli o problemi che inevitabilmente possono sorgere durante le fasi di progettazione, quindi può essere necessario in alcuni casi riadattare improvvisamente il modello al fine di renderlo coerente con gli obiettivi prefissati.

### 1.1 Purpose of the system

Come già visto nel Requirements Analysis Document, l'obiettivo del Sistema è quello di gestire la sezione dipendenti di un'azienda di servizi al cittadino. In modo particolare, il Sistema avrà l'onere di gestire in modo automatico tutte le informazioni riguardanti i dipendenti dell'azienda in questione, gli stipendi e le turnazioni trimestrali di questi ultimi. I dipendenti sono tutti full-time, ma svolgono quattro servizi differenti.

### 1.2 Design goals

Il nostro obiettivo principale è quello di realizzare un sistema performante e affidabile che permetta al datore e ai dipendenti dell'azienda di svolgere le funzionalità previste in fase di analisi e di gestire in maniera efficiente ed efficace i dati necessari allo svolgimento delle funzionalità stesse. Vogliamo inoltre che il sistema sia facilmente manutenibile e facilmente personalizzabile dalla specifica azienda che userà il software in futuro, senza compromettere l'architettura già esistente.

### 1.3 Definitions, acronyms, and abbreviations

Come già visto nel Requirements Analysis Document, riportiamo nella tabella di seguito definizioni, acronimi e abbreviazioni in modo da rendere anche il System Design Document uniforme e meno ambiguo.

Termine	Significato
Utente	Generico individuo autenticato che utilizza il Sistema; può essere un Dipendente (di tipo 1, 2, 3 o 4) o il Datore Di Lavoro.
Dipendente	Utilizzatore autenticato dal Sistema che lavora nell'azienda; può essere di quattro tipi (1, 2, 3 o 4) a seconda del servizio assegnatogli e/o svolto.
Datore Di Lavoro	Utilizzatore autenticato dal Sistema che supervisiona il corretto funzionamento dell'azienda e dell'operato dei dipendenti. Abbreviato in "DDL".
Credenziali Login	Insieme di dati necessari all'Utente per autenticarsi e utilizzare il Sistema. Nello specifico sono: Matricola, Password.

Credenziali Entrata/Uscita	Insieme di dati necessari all'Utente per autenticarsi e segnalare la propria entrata/uscita durante un turno ben definito. Nello specifico sono: Matricola, Nome, Cognome. Entrata e uscita vengono normalmente segnalate da un Terminale presente all'ingresso dell'azienda. Tuttavia, l'entrata può essere segnalata dal Dipendente dal Software dell'azienda qualora egli sia in ritardo e il Terminale risulti quindi indisponibile.
Terminale	Dispositivo presente all'ingresso dell'azienda, attraverso il quale i dipendenti possono effettuare la rilevazione dell'entrata e dell'uscita dal turno previsto dello stesso.
Account	Insieme di dati che identificano un Utente autenticato all'interno del Sistema durante una sessione di utilizzo.
Stato del Dipendente Entrata	Fornisce informazioni sul Dipendente rispetto all'ingresso del turno che deve svolgere. Di seguito i possibili stati di un'entrata: Presente, Presente Ritardo, Assente.
Stato del Dipendente Uscita	Fornisce informazioni sul Dipendente rispetto all'uscita del turno che si è appena svolto. Di seguito i possibili stati di un'uscita: Manuale, Automatica.
DBMS (DataBase Management System)	Sistema che organizza in modo strutturato tutti i dati necessari al funzionamento dell'Azienda e fornisce una serie di funzionalità per creare, leggere, aggiornare e rimuovere i dati.

## 1.4 References

Fonti utili usate durante la scrittura del System Design Document:

- *Object-Oriented Software Engineering: Using Uml, Patterns and Java* di Bernd Bruegge e Allen H. Dutoit.
- *Software Engineering* di Ian Sommerville.
- *Basi di Dati* di Paolo Atzeni, Stefano Ceri, Piero Fraternali, Stefano Paraboschi e Riccardo Torlone.
- MySQL 8.0 Reference Manual: <https://dev.mysql.com/doc/refman/8.0/en/>

## 1.5 Overview

Già dalle specifiche fornite dal cliente appare chiaro che sono coinvolti più dispositivi, in particolare un paio di postazioni presenti all'ingresso dell'azienda (esclusive per la rilevazione delle entrate e delle uscite) e di conseguenza altri dispositivi diversi dai precedenti usati dai dipendenti e dal datore per svolgere le funzionalità di cui necessitano. Inoltre si evince che il sistema dovrà gestire un elevato volume di dati che cresceranno esponenzialmente all'aumentare dei dipendenti e che verranno archiviati per lunghi intervalli di tempo: turni, stipendi, presenze, variazioni, comunicazioni e così via. Occorre quindi un'architettura che sia in grado di gestire nel modo migliore possibile lo scenario appena descritto.

## 2. Current software architecture

Supponiamo che non esista nessuna architettura presente in funzione nell'azienda.

### 3. Proposed software architecture

Discutiamo di seguito tutti i dettagli dell'architettura proposta.

#### 3.1 Overview

Alla luce delle prime considerazioni emerse nella parte introduttiva e analizzando le principali architetture esistenti e comunemente usate, la nostra scelta è ricaduta su un'architettura di tipo Repository. L'architettura in questione gode sicuramente di svariati punti di forza:

- Gestisce in maniera efficiente grandi moli di dati.
- Permette di astrarre il funzionamento interno dei sottosistemi o di aggiungerne nuovi senza troppe preoccupazioni.
- Gestisce in maniera centralizzata i dati e di conseguenza tutte le operazioni a essi connesse come operazioni di backup e di ripristino.

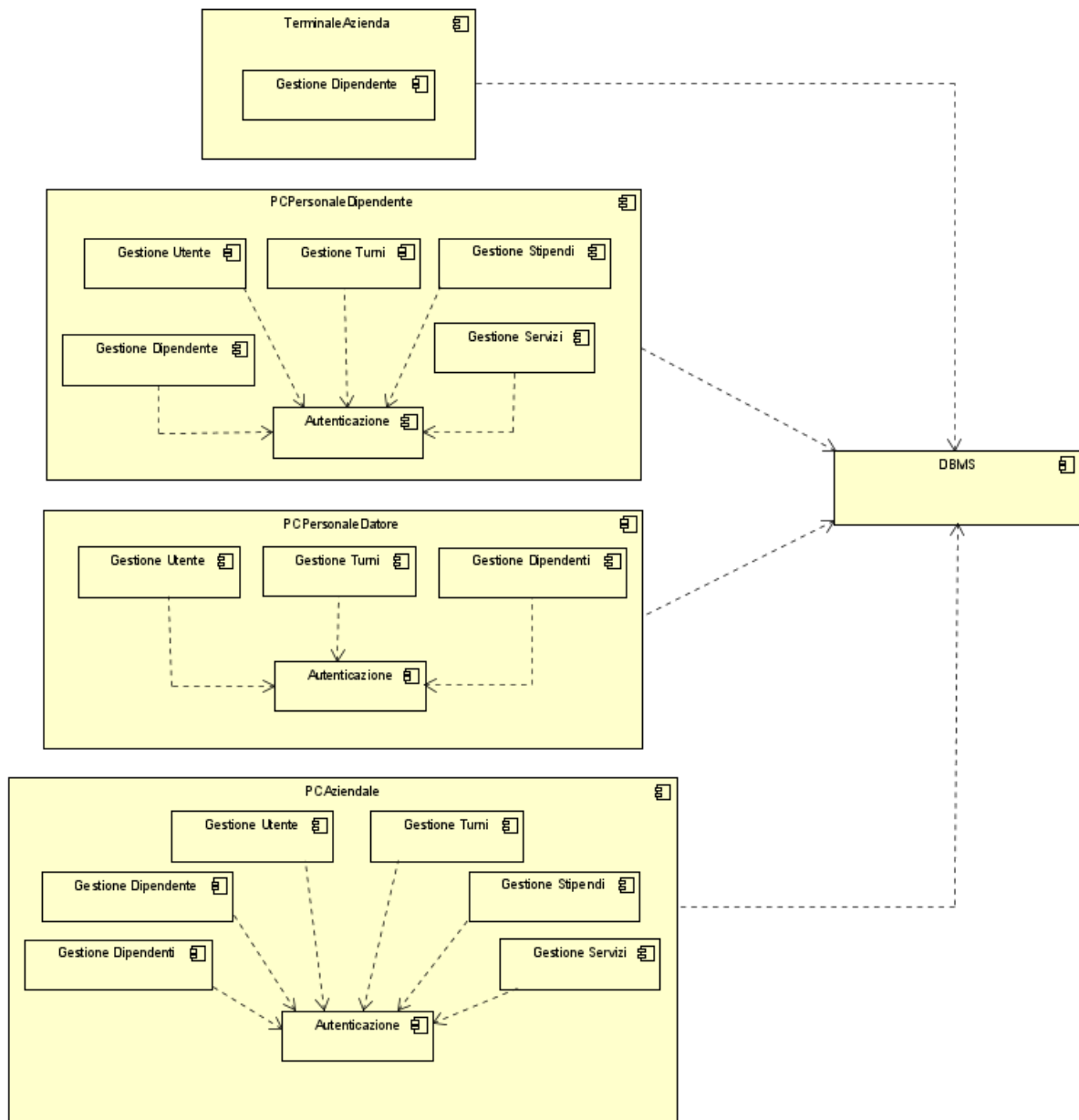
Tuttavia come già anticipato nell'introduzione tutte le scelte portano inevitabilmente a dei compromessi. A fianco dei vantaggi dell'architettura scelta dobbiamo tenere in considerazione alcuni svantaggi:

- Tutti i sottosistemi devono adattarsi a un modello comune per potersi interfacciare con il repository. Di conseguenza una volta scelto il modello non è possibile cambiarlo radicalmente poiché renderebbe necessaria la modifica del repository e di tutti i sottosistemi.
- Un guasto al repository propaga inevitabilmente i disagi all'intera azienda.
- Utilizzare il repository su un sistema distribuito non è semplice perché bisogna mantenere sempre la coerenza dei dati (ed evitare cioè problemi di consistenza o di ridondanza).
- Il repository se non opportunamente dimensionato può diventare un collo di bottiglia per l'intero sistema.

Non sempre questi svantaggi costituiscono un limite, ma è fondamentale conoscerli in modo da progettare al meglio i singoli sottosistemi.

#### 3.2 Subsystem decomposition

Riportiamo di seguito il *component diagram* in cui si evince la decomposizione in sottosistemi:

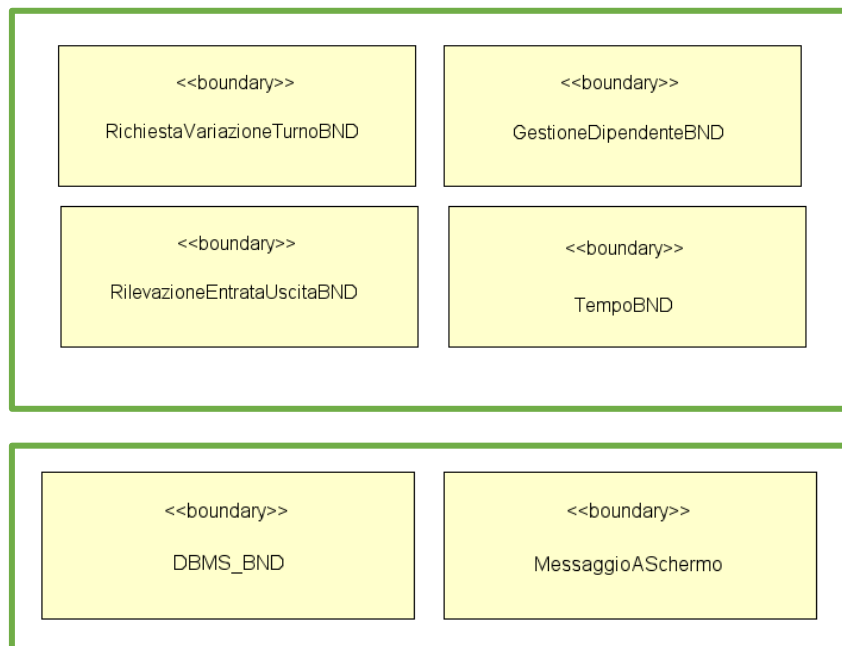


Riportiamo di seguito maggiori dettagli su come gli oggetti sono suddivisi all'interno dei sottosistemi:

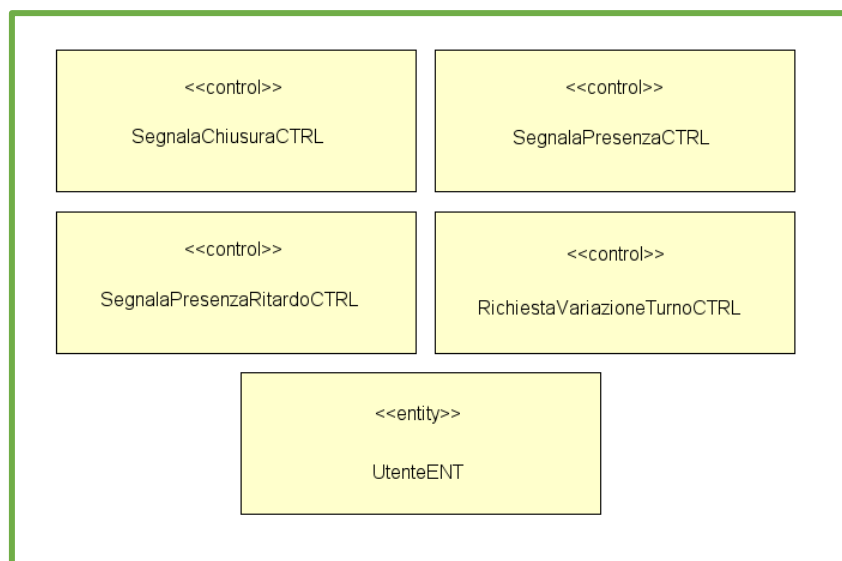
- **TerminaleAzienda**

### **Gestione Dipendente**

Boundary:



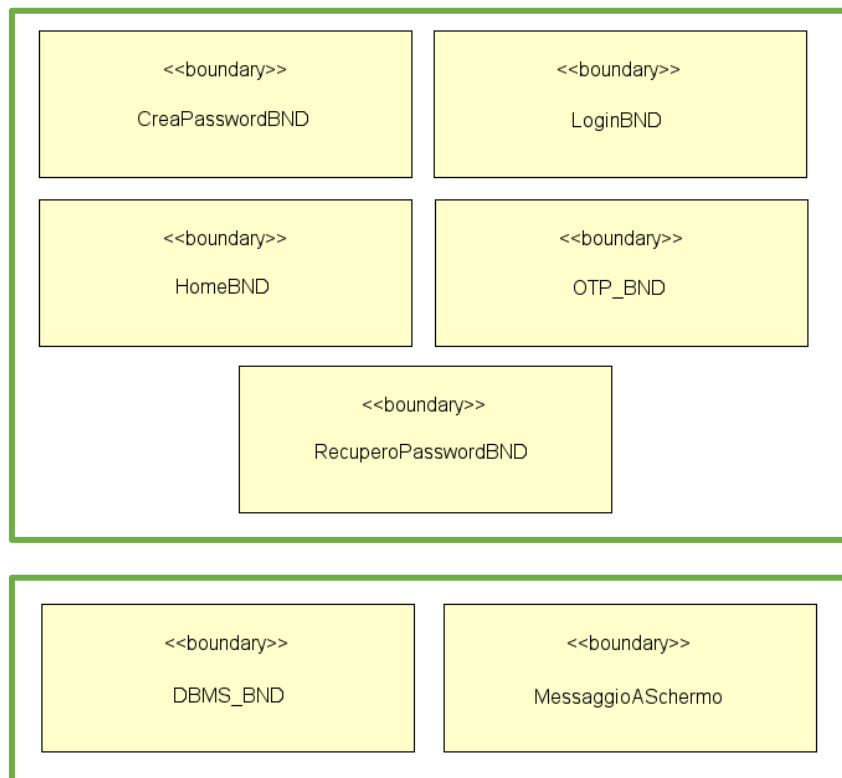
Control/Entity:



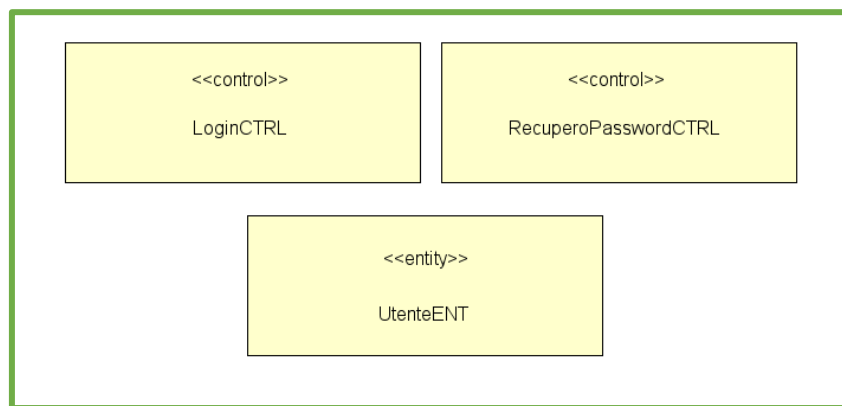
- **PCPersonaleDipendente**

**Autenticazione**

Boundary:



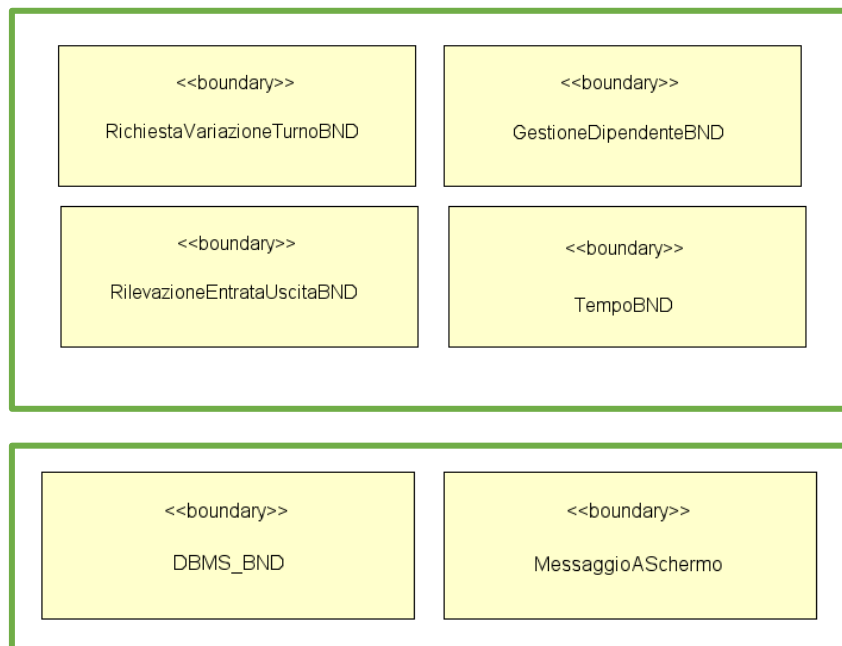
Control/Entity:



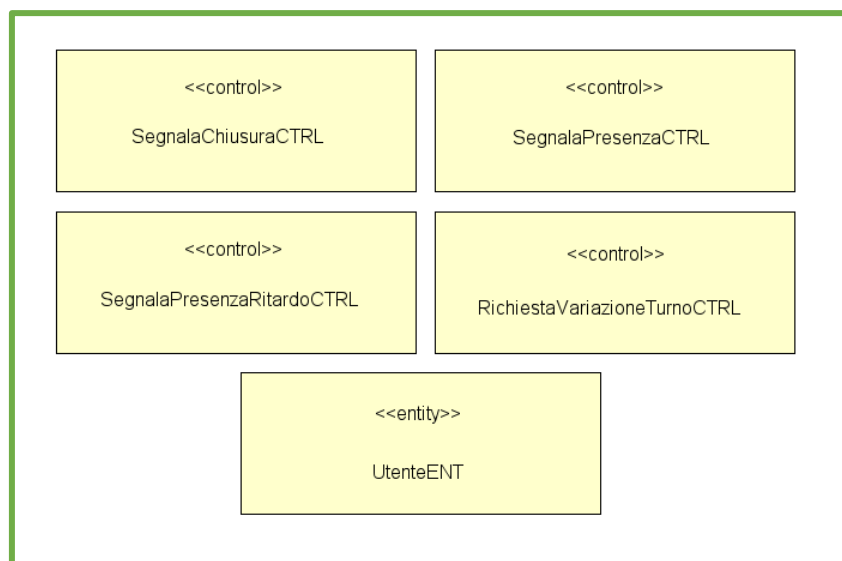
## Gestione Dipendente

Boundary:



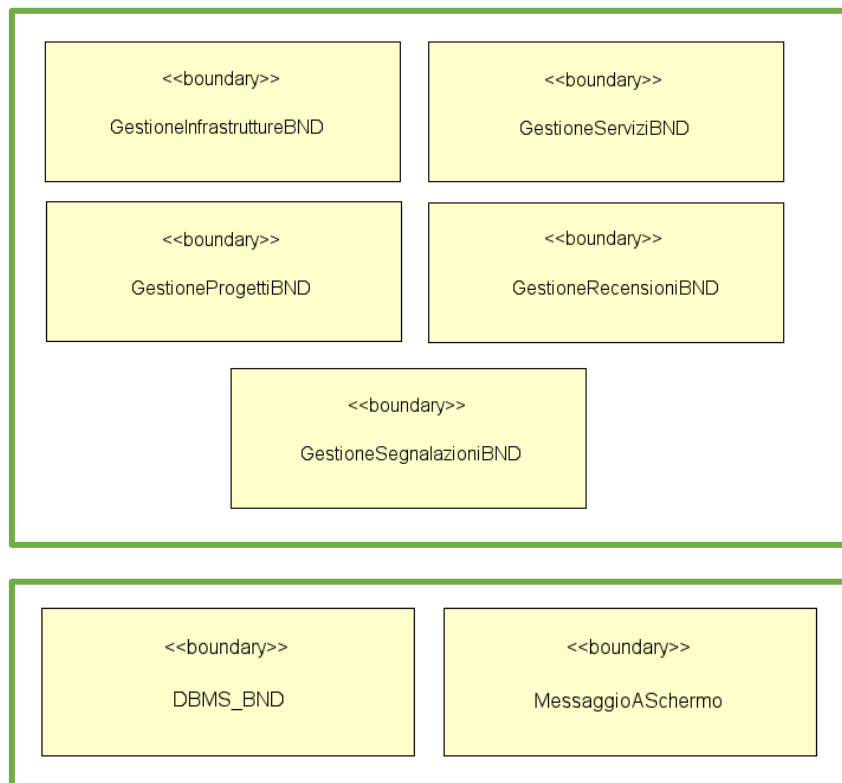


Control/Entity:

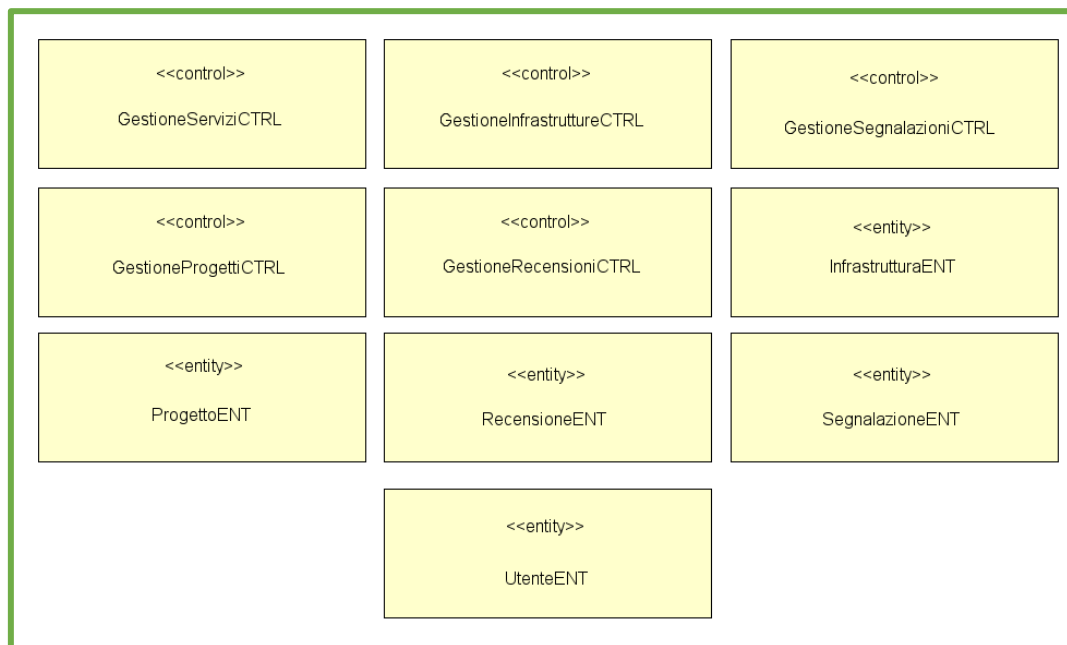


## Gestione Servizi

Boundary:

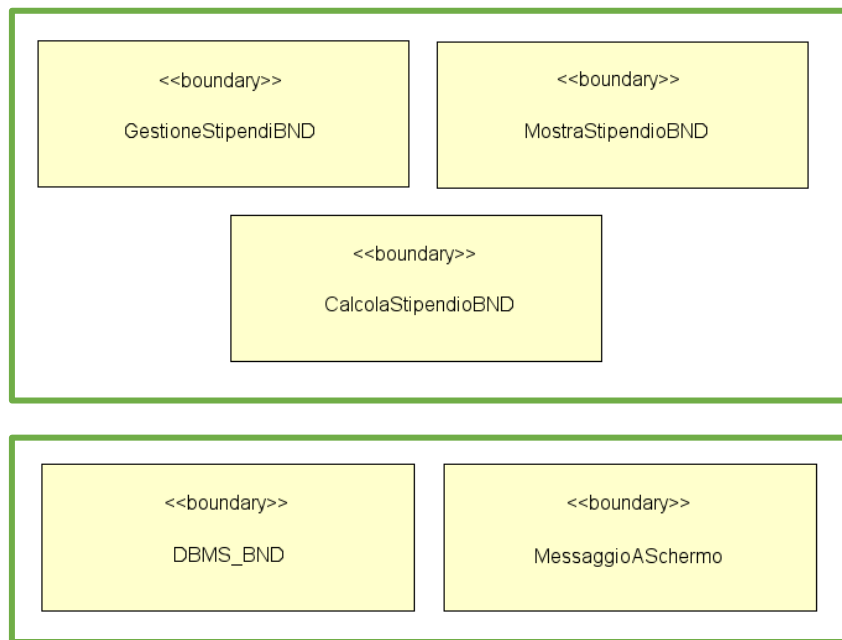


Control/Entity:

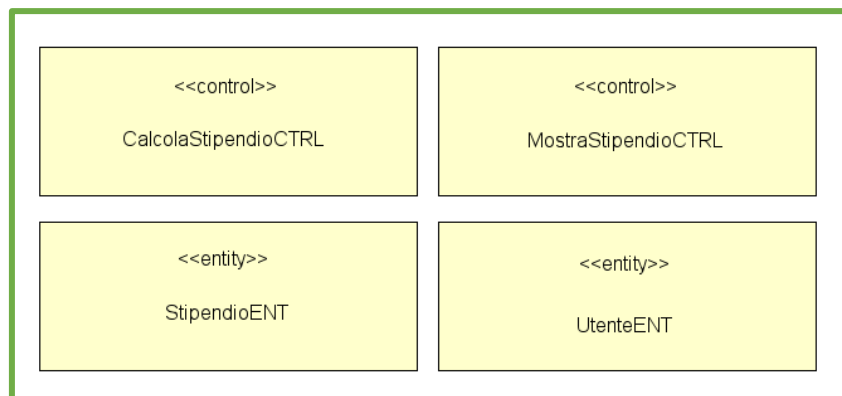


## Gestione Stipendi

Boundary:

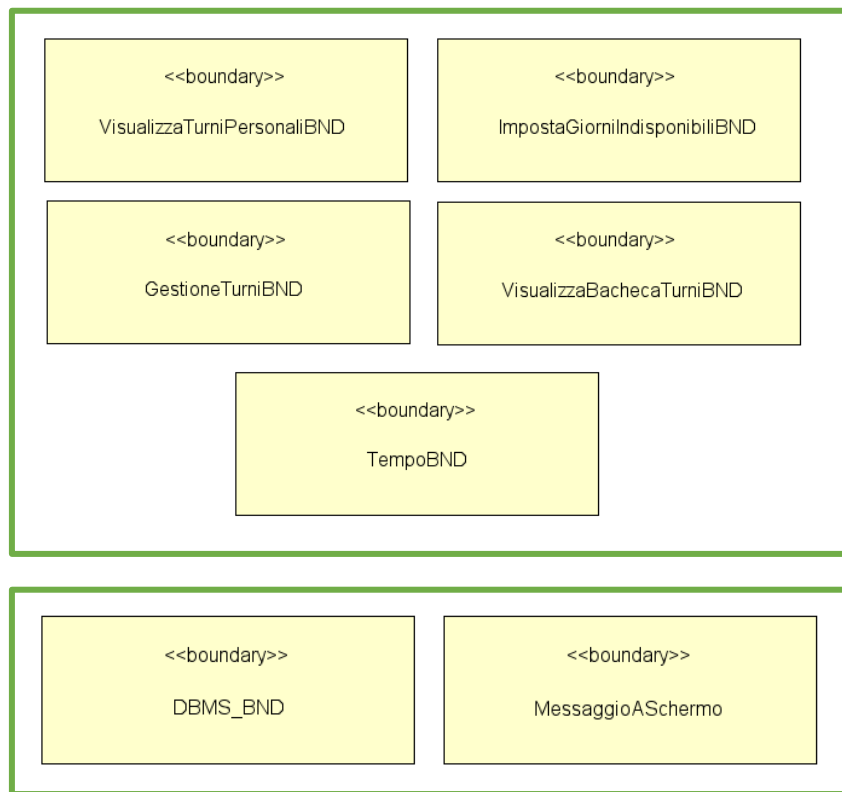


Control/Entity:

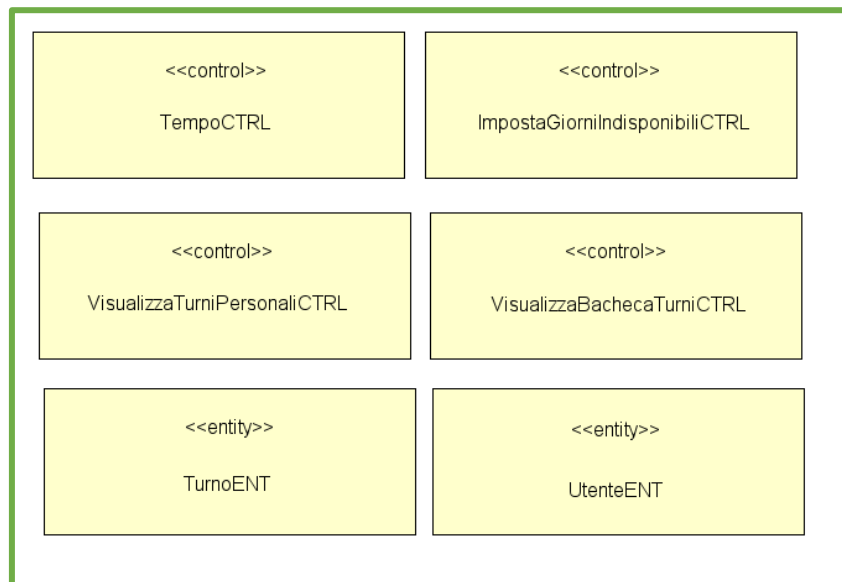


**Gestione Turni**

Boundary:

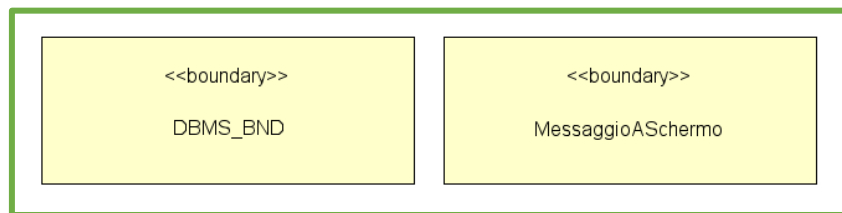
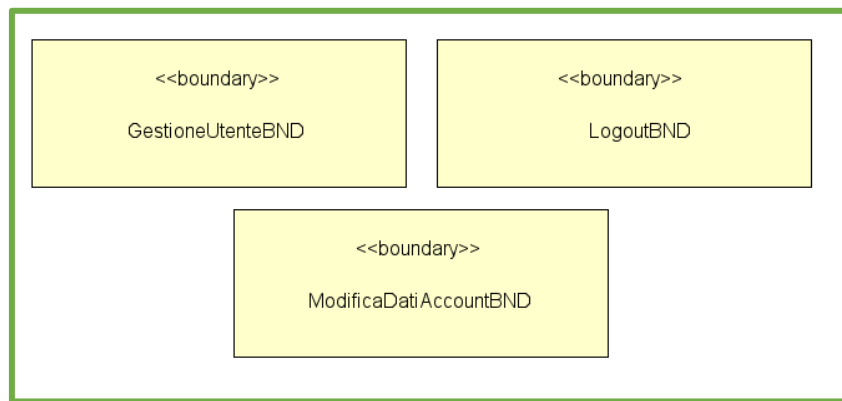


Control/Entity:

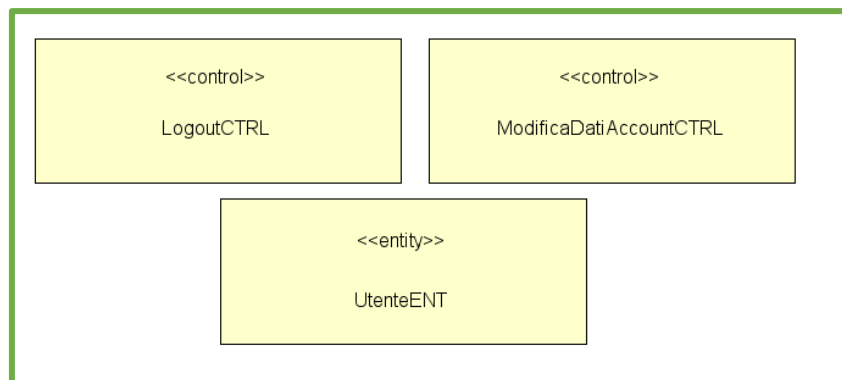


## Gestione Utente

Boundary:



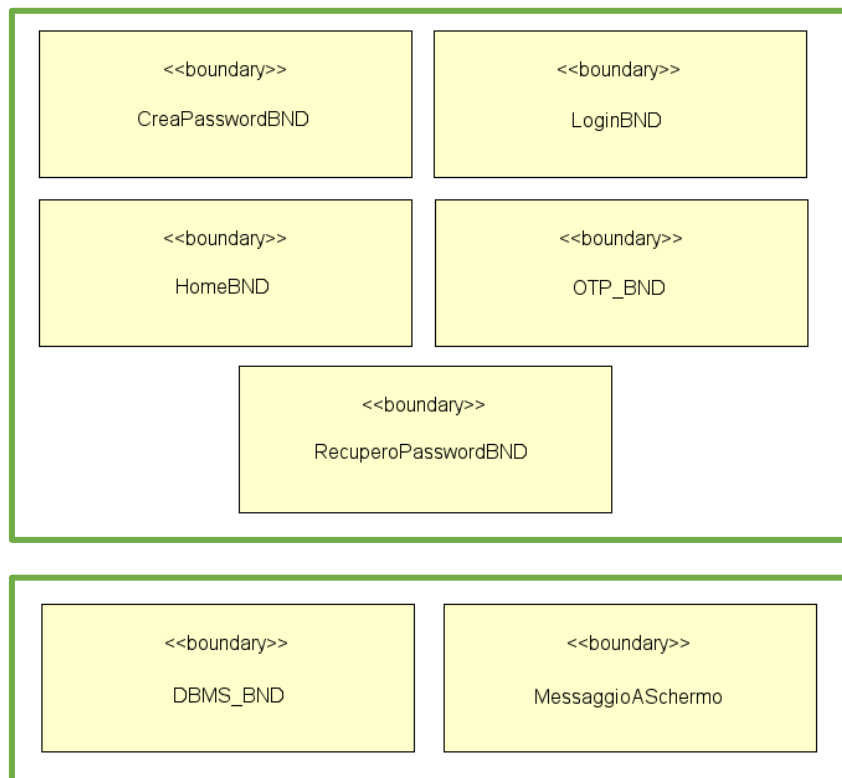
Control/Entity:



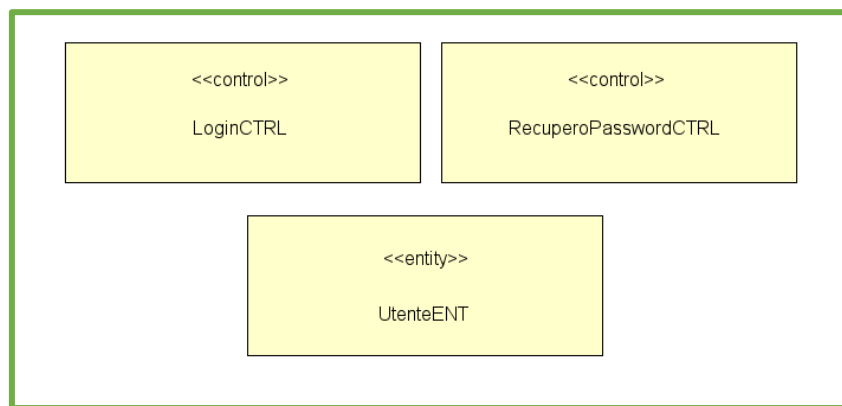
- **PC Personale Datore**

## Autenticazione

Boundary:



Control/Entity:

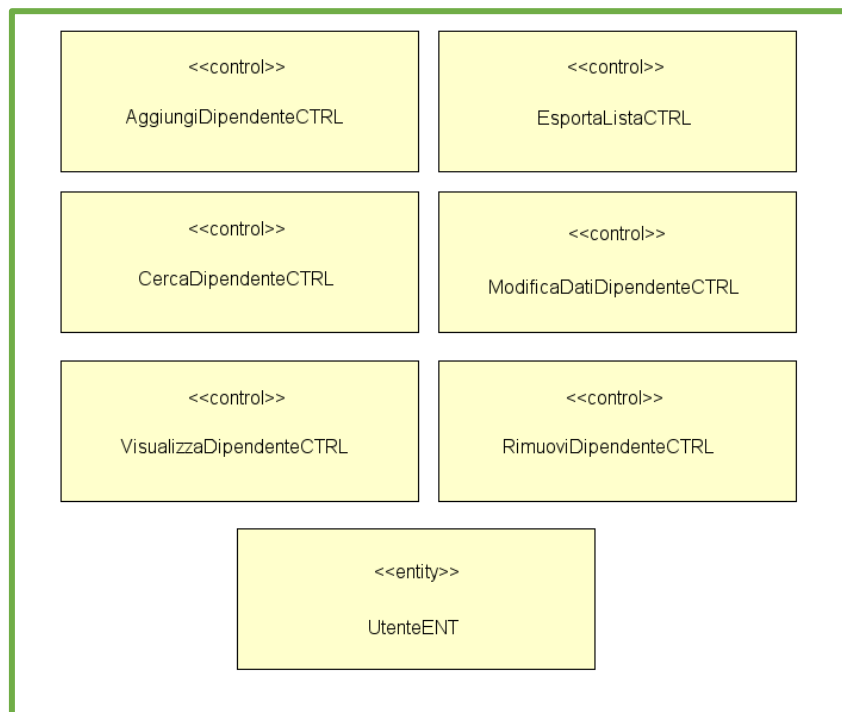


## Gestione Dipendenti

Boundary:

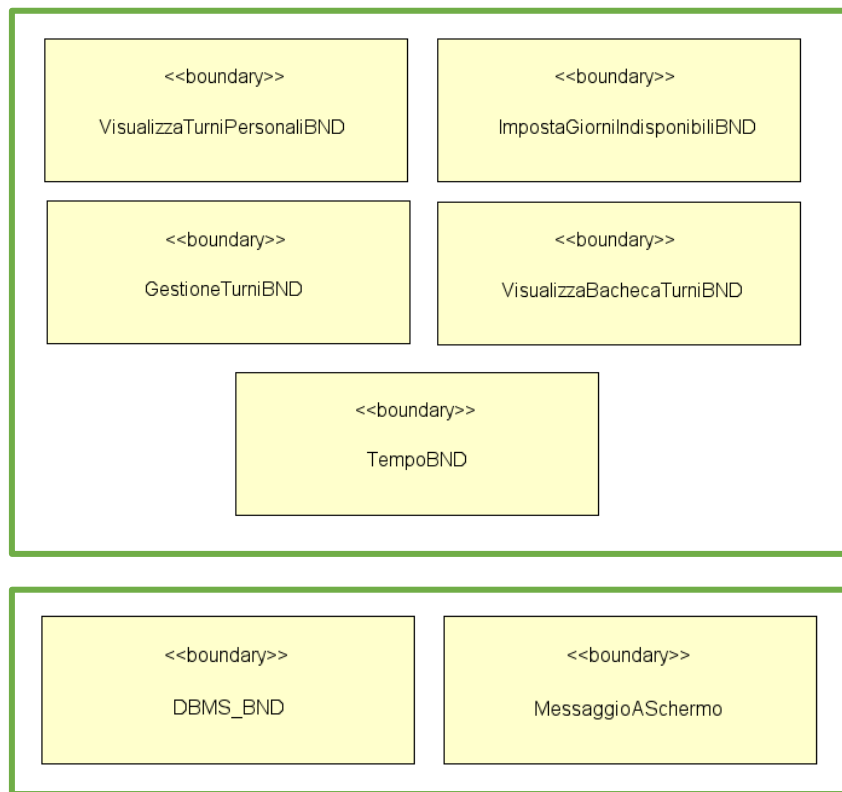


Control/Entity:

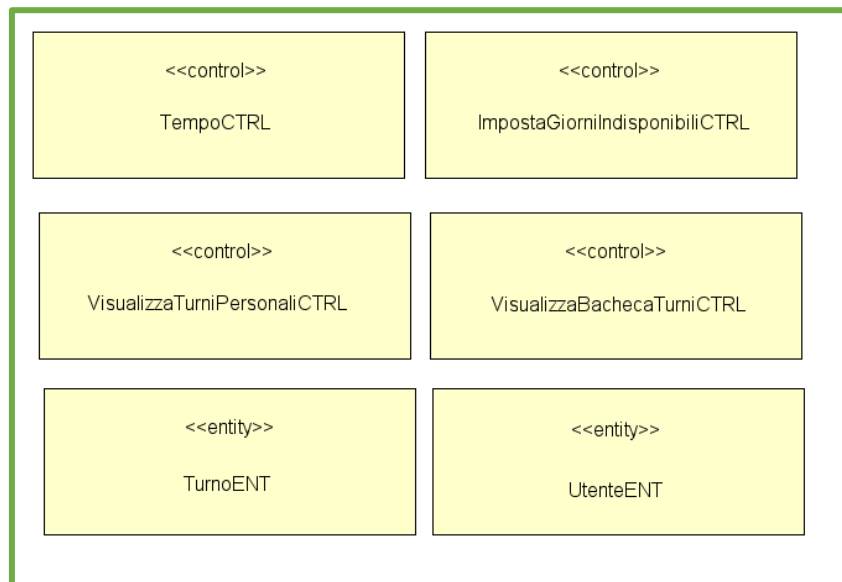


**Gestione Turni**

Boundary:



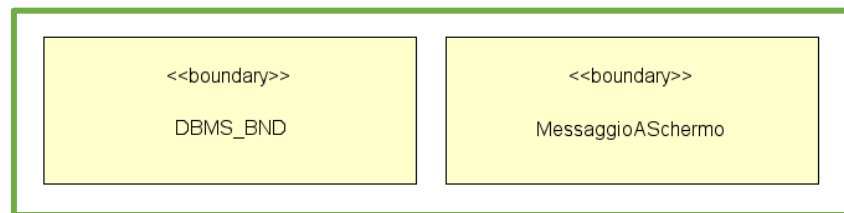
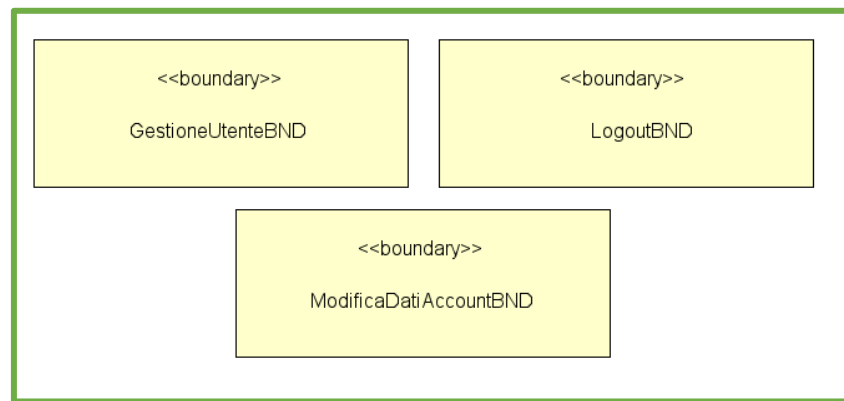
Control/Entity:



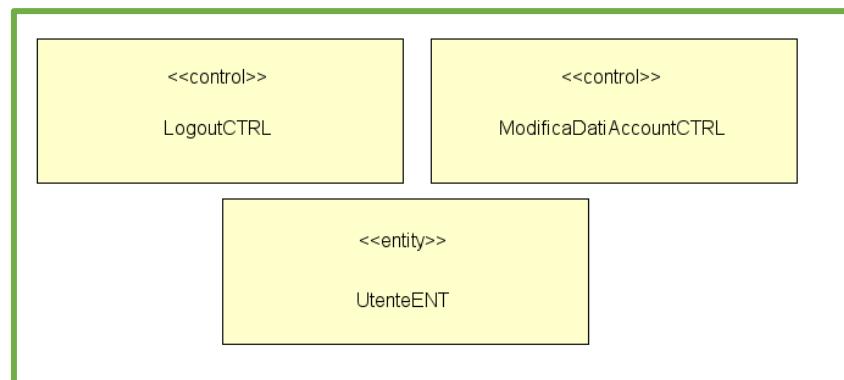
## Gestione Utente

Boundary:





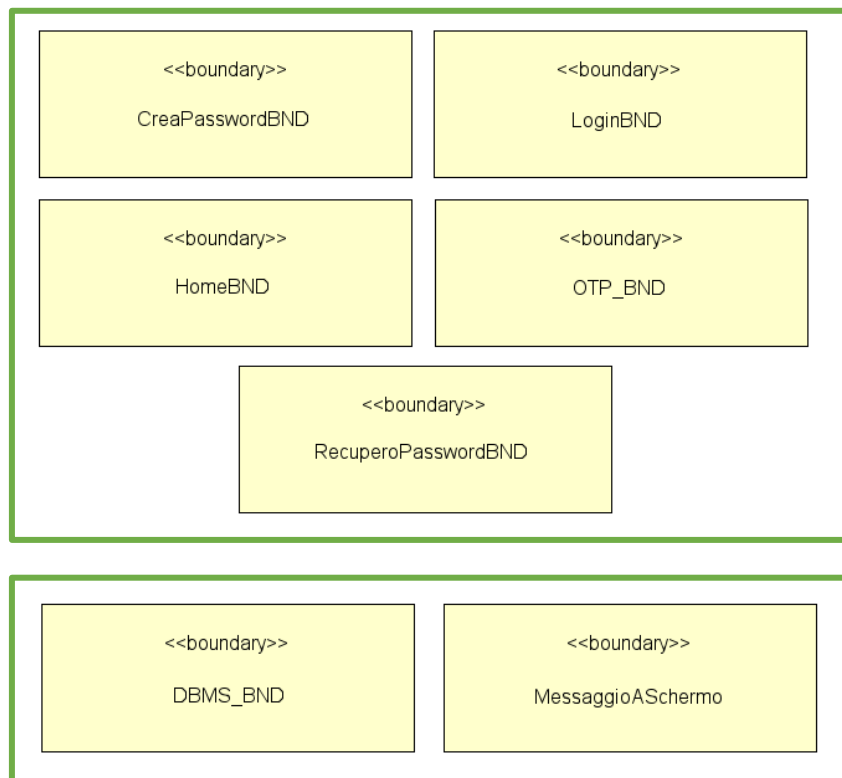
Control/Entity:



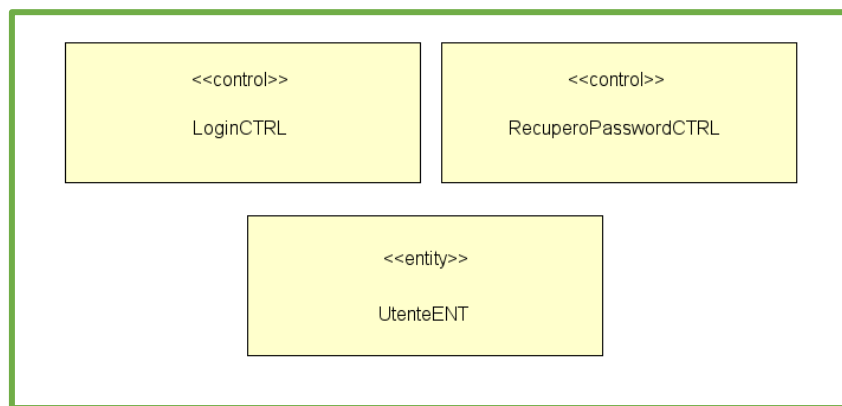
- **PCAziendale**

## Autenticazione

Boundary:

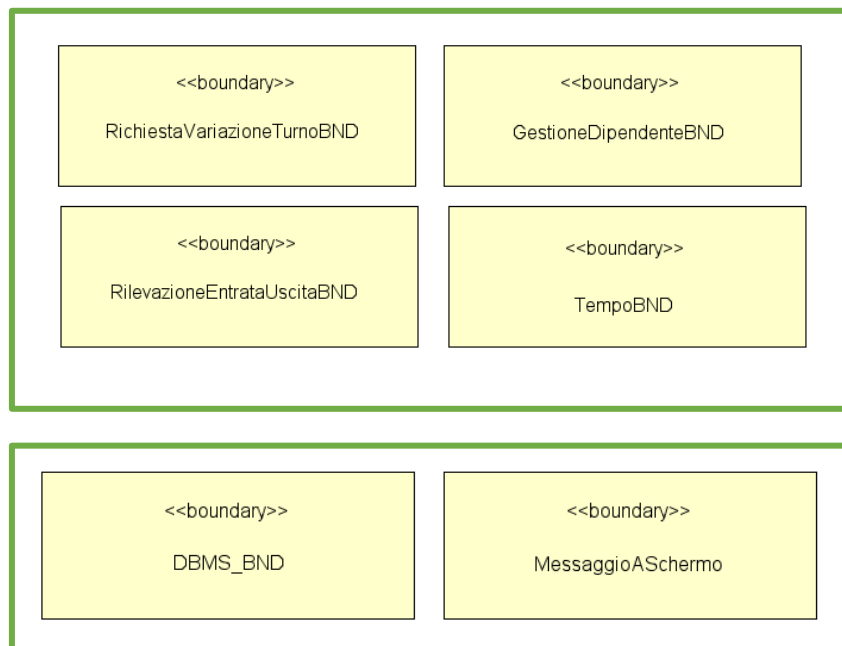


Control/Entity:

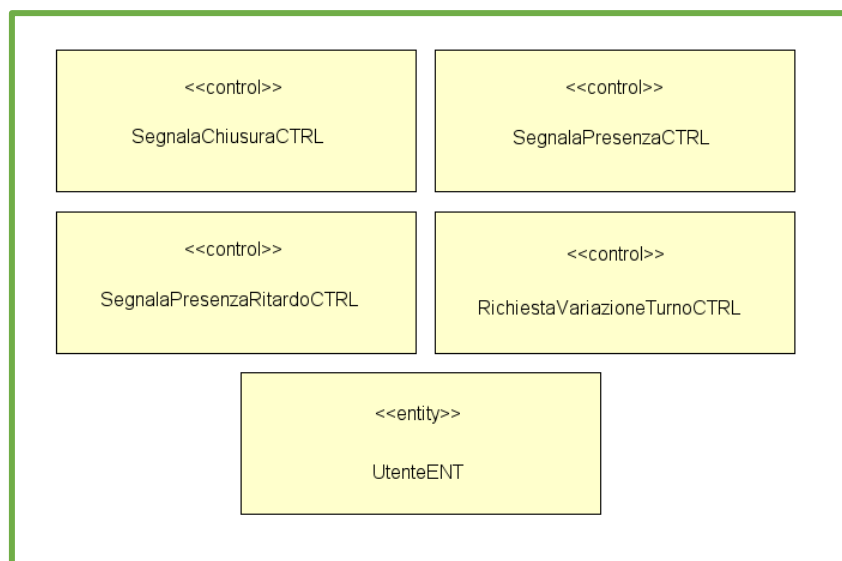


## Gestione Dipendente

Boundary:



Control/Entity:

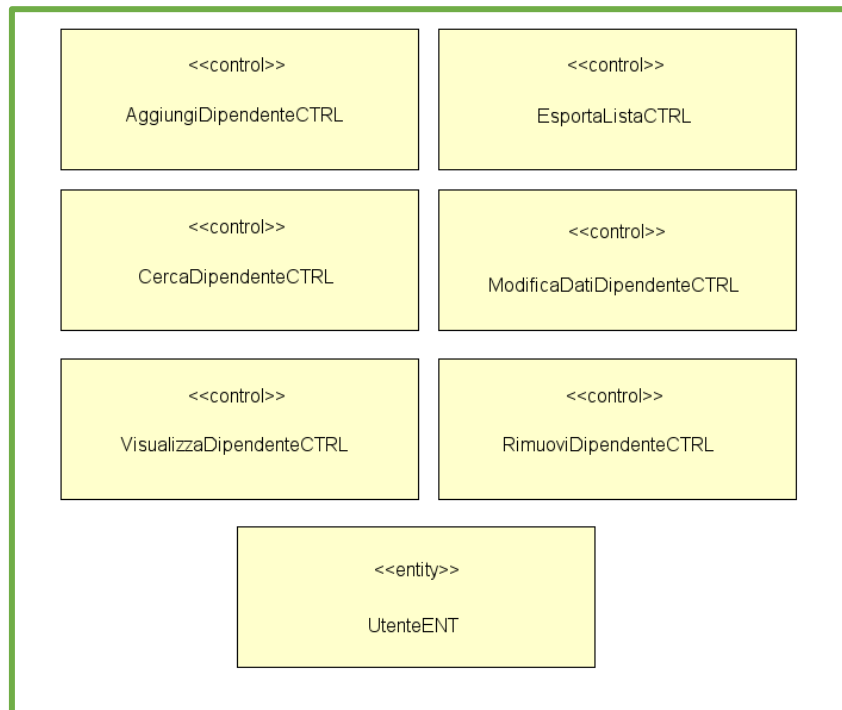


## Gestione Dipendenti

Boundary:

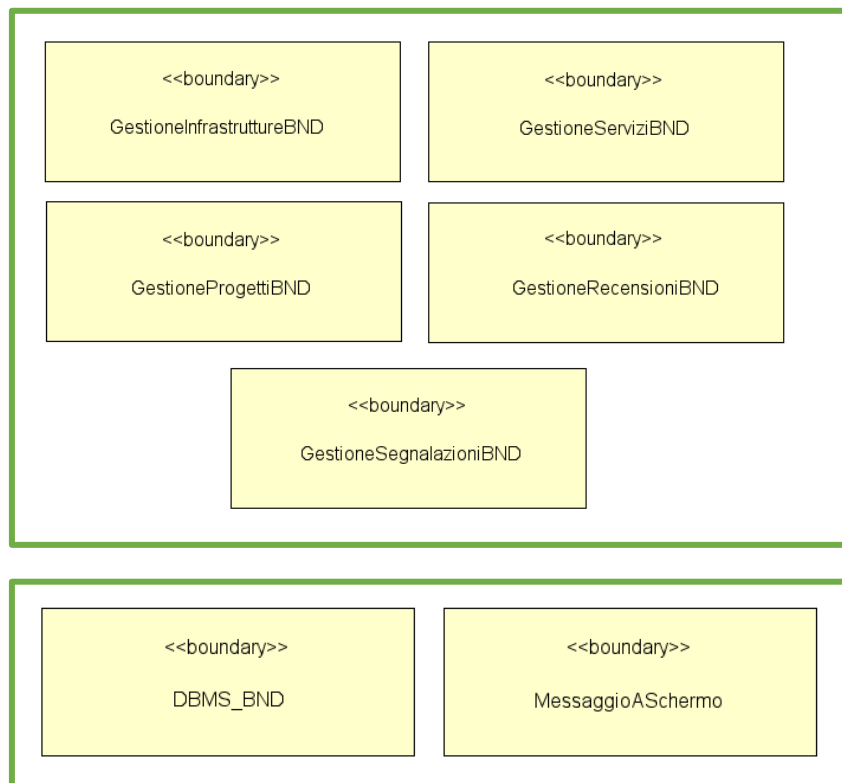


Control/Entity:

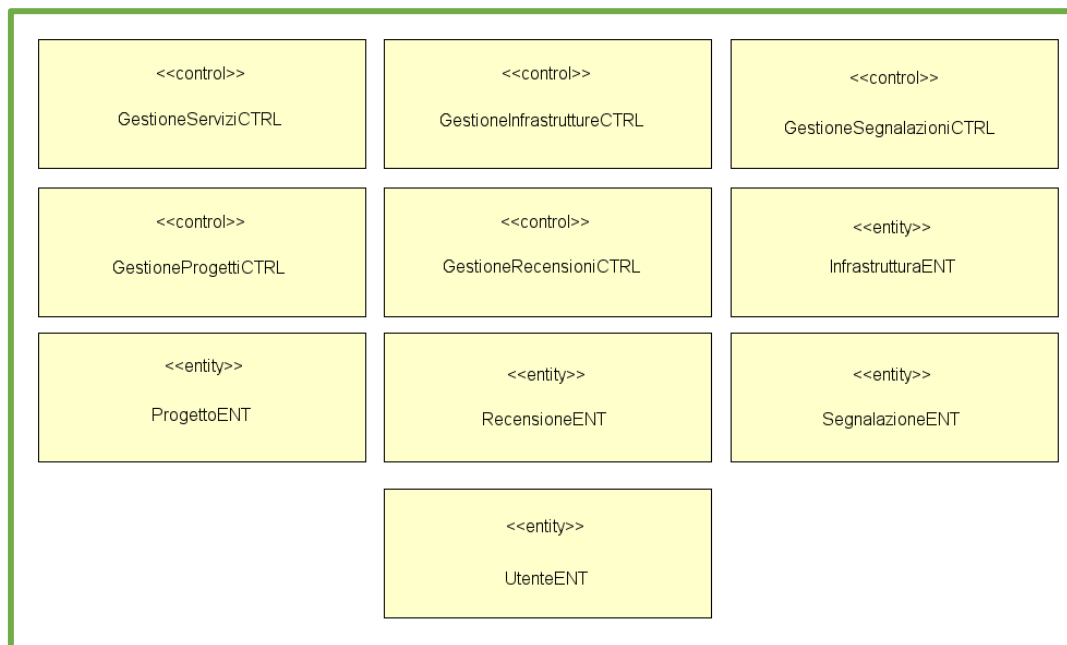


**Gestione Servizi**

Boundary:

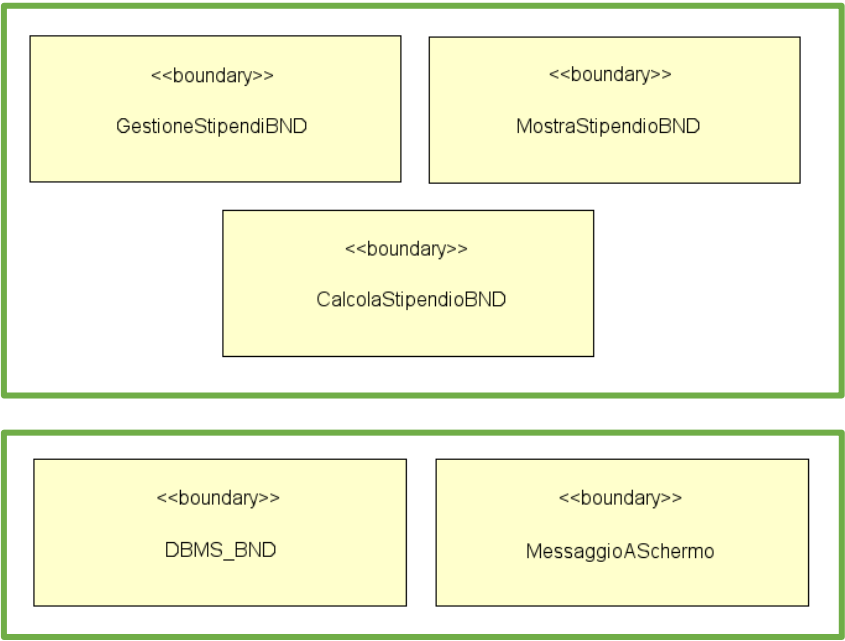


Control/Entity:

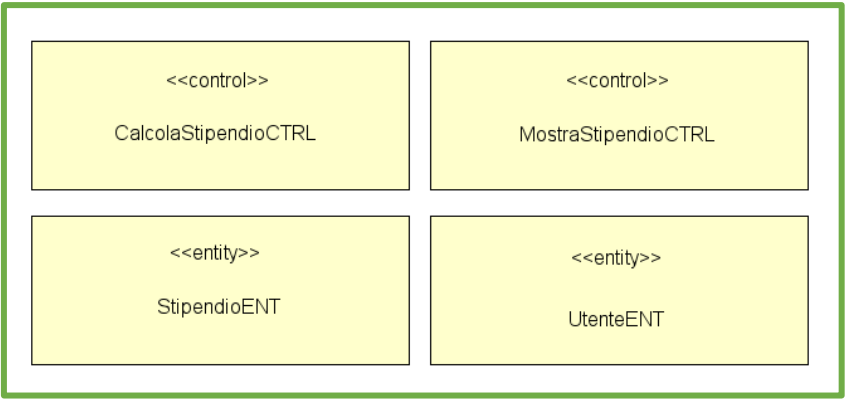


## Gestione Stipendi

Boundary:

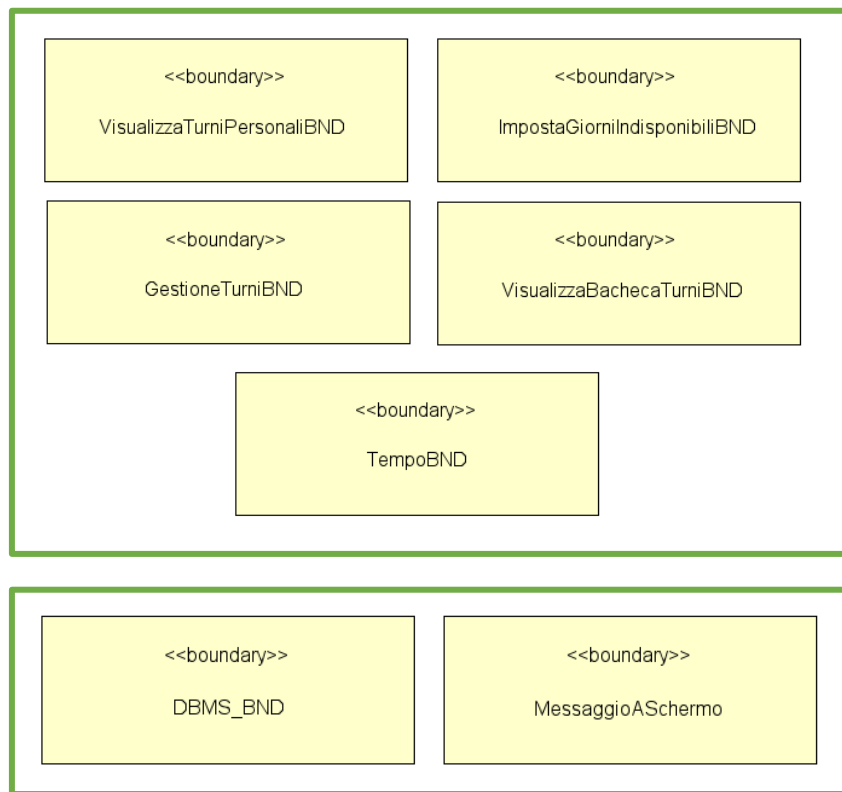


Control/Entity:



Gestione Turni

Boundary:

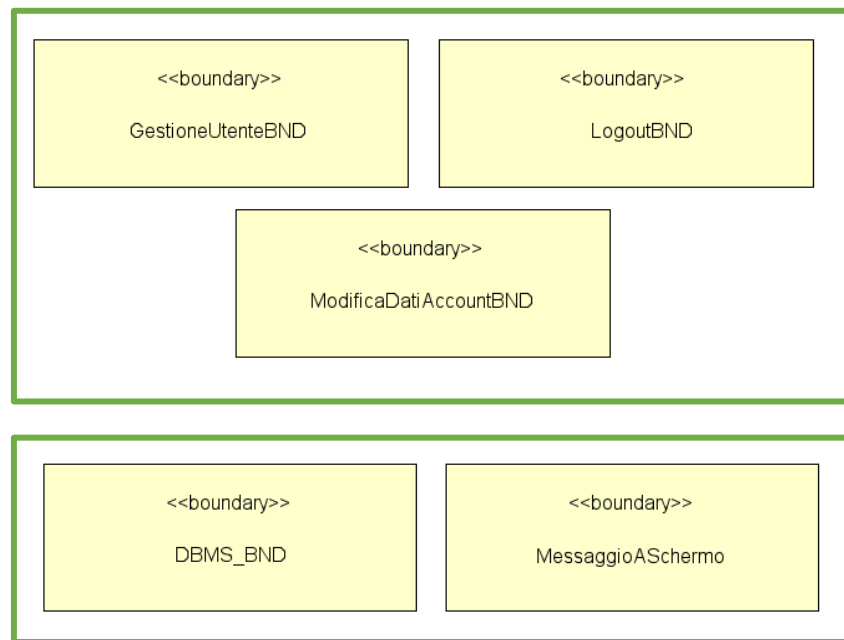


Control/Entity:

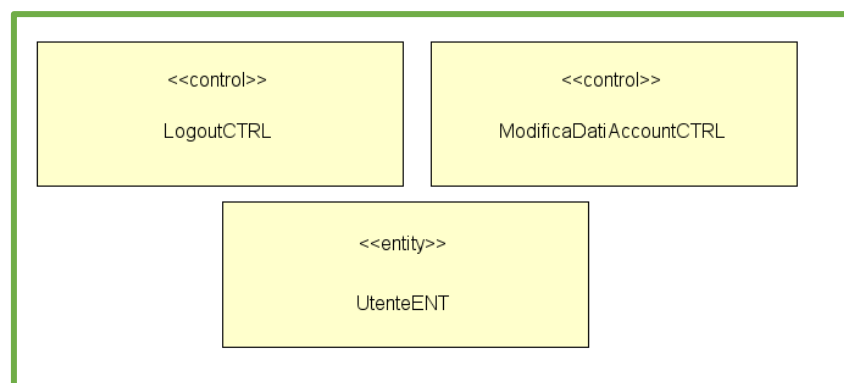


**Gestione Utente**

Boundary:



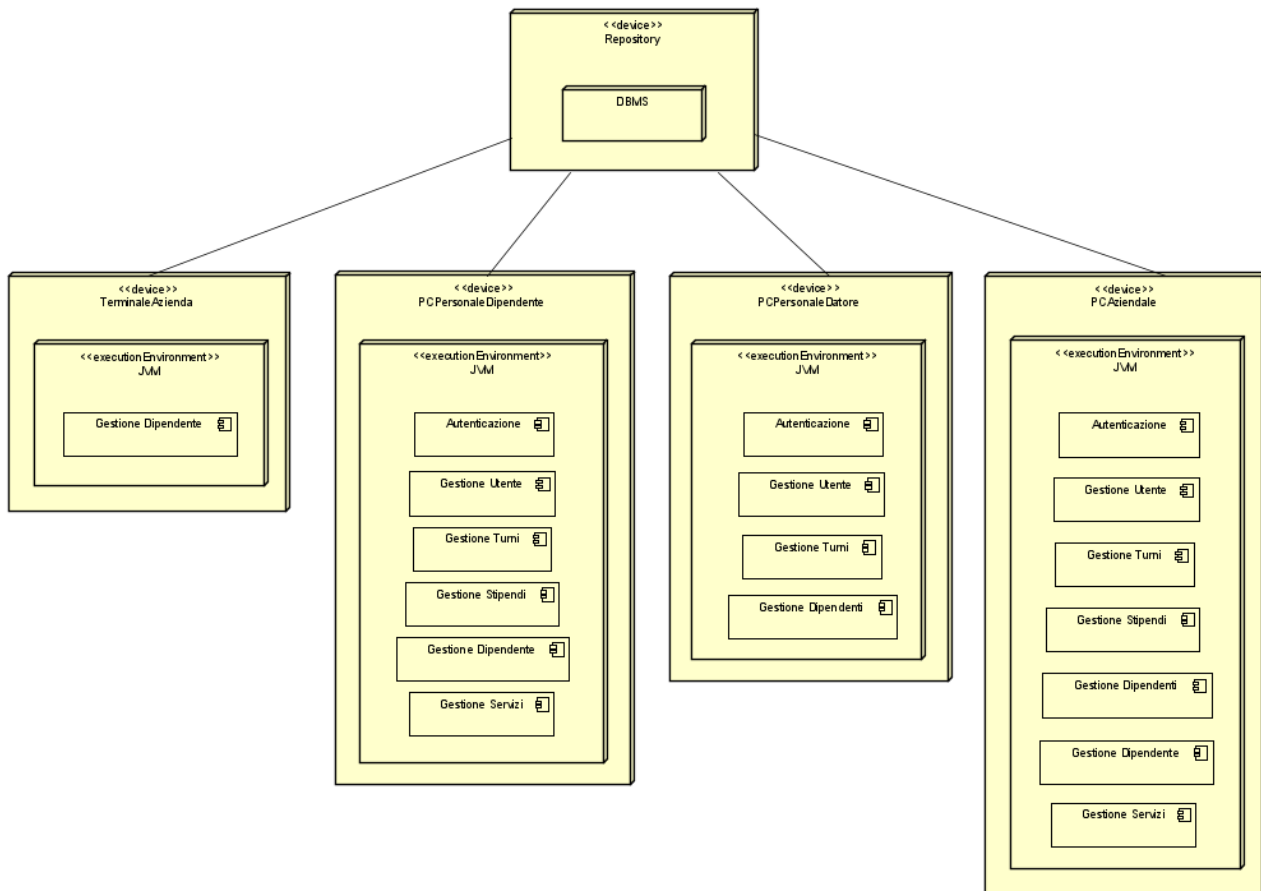
Control/Entity:



### 3.3 Hardware/software mapping

Seguendo il modello repository, è stata eseguita la mappatura e riportata nel seguente *deployment diagram*:





Abbiamo previsto cinque nodi e su ciascuno sono installati i pacchetti necessari per le varie funzionalità:

- Il nodo *Repository* ospita il DBMS.
- Il nodo *TerminaleAzienda* permette ai dipendenti di segnalare la loro entrata e la loro uscita.
- Il nodo *PCPersonaleDipendente* permette ai dipendenti di svolgere le funzionalità per loro previste. Alcune di esse sono disponibili soltanto durante i turni previsti.
- Il nodo *PCPersonaleDatore* permette al datore di svolgere le funzionalità per lui previste.
- Il nodo *PCAziendale* è liberamente fruibile dai dipendenti presenti in azienda in quel momento oppure dal datore.

Si suppone che il nodo *Repository* sia attivo 24 ore su 24 in quanto necessario al funzionamento dell'intero sistema.

Si suppone inoltre che il nodo *TerminaleAzienda* sia attivo 24 ore su 24 per permettere la rilevazione delle entrate e delle uscite dei dipendenti.

Si suppone infine che il nodo *PCAziendale* sia attivo 24 ore su 24 per permettere l'esecuzione automatica delle seguenti due funzionalità:

- Calcola Stipendio Mensile
  - 1) 01/01 alle 01:10:00 (stipendi di dicembre)

- 2) 01/02 alle 01:10:00 (stipendi di gennaio)
  - 3) 01/03 alle 01:10:00 (stipendi di febbraio)
  - 4) 01/04 alle 01:10:00 (stipendi di marzo)
  - 5) 01/05 alle 01:10:00 (stipendi di aprile)
  - 6) 01/06 alle 01:10:00 (stipendi di maggio)
  - 7) 01/07 alle 01:10:00 (stipendi di giugno)
  - 8) 01/08 alle 01:10:00 (stipendi di luglio)
  - 9) 01/09 alle 01:10:00 (stipendi di agosto)
  - 10) 01/10 alle 01:10:00 (stipendi di settembre)
  - 11) 01/11 alle 01:10:00 (stipendi di ottobre)
  - 12) 01/12 alle 01:10:00 (stipendi di novembre)
- Genera Turni
    - 1) 01/03 alle 00:00:00 (turni trimestre aprile-giugno)
    - 2) 01/06 alle 00:00:00 (turni trimestre luglio-settembre)
    - 3) 01/09 alle 00:00:00 (turni trimestre ottobre-dicembre)
    - 4) 01/12 alle 00:00:00 (turni trimestre gennaio-marzo)

Un file di configurazione (testo) presente nella cartella del software permette di capire da quale nodo si sta eseguendo il software, in modo da mostrare le schermate corrette e assicurarsi che queste due funzionalità vengano eseguite solo dal nodo corretto.

### 3.4 Persistent data management

Per la gestione dei dati persistenti necessari al corretto funzionamento del progetto si è scelto di implementare mediante **Google Cloud Platform** (ma è anche possibile effettuare delle prove su un database locale) un DBMS relazionale e in particolare è stato usato MySQL. Non è necessario salvare dati su file locali o serializzare oggetti.

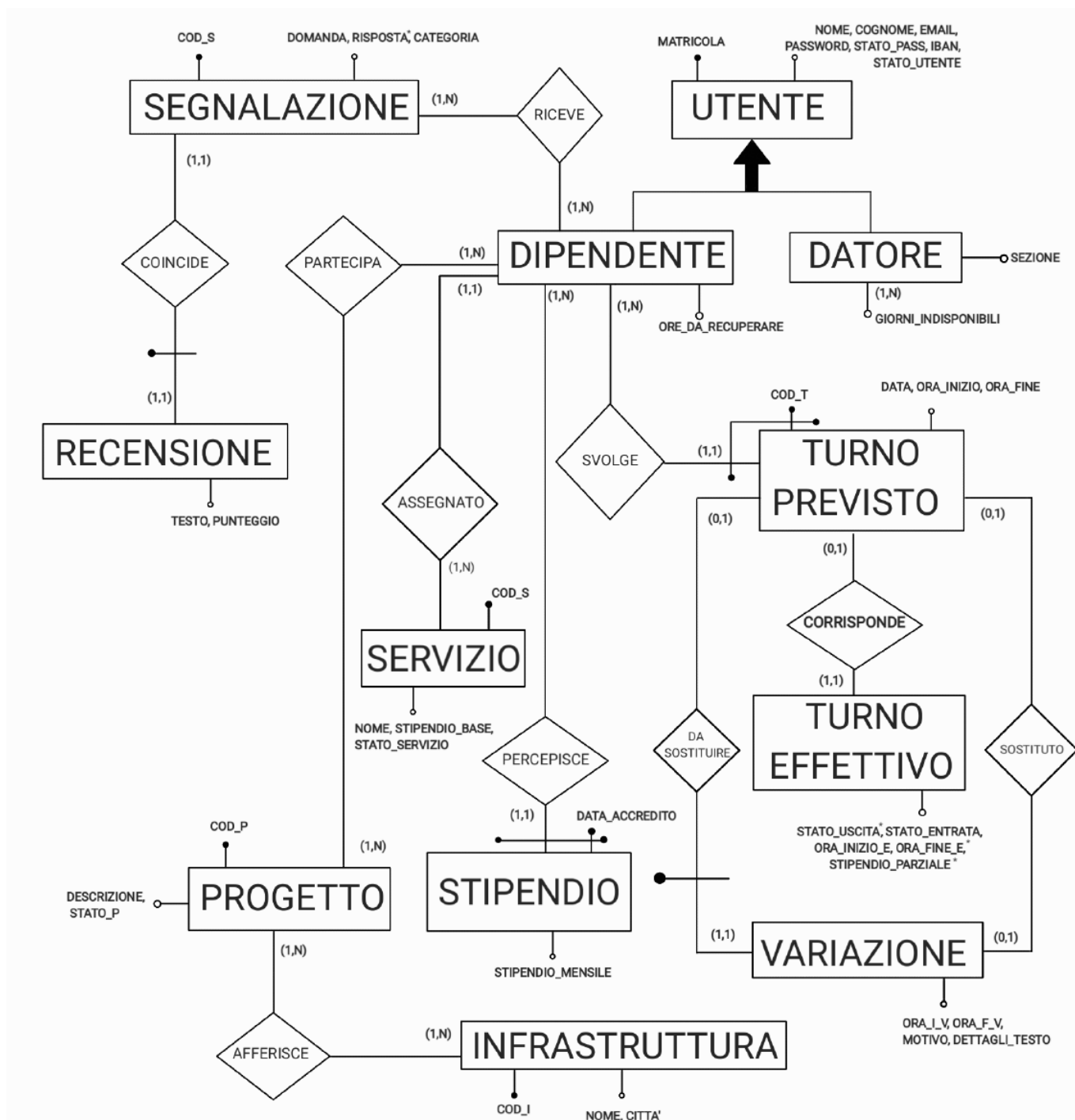
#### 3.4.1 Progettazione Modello E-R

Sulla base delle specifiche del progetto, sono state identificate le entità e le relazioni necessarie alla memorizzazione dei dati, con relativi attributi.

È stata posta particolare attenzione alla scelta di nomi auto-esplicativi. Inoltre per evitare conflitti di nomi con alcune parole riservate in MySQL, l'attributo per memorizzare la password in utente viene chiamato `pass` e si evita l'utilizzo della parola `data`.

Una piccola precisazione: il modello presenta un'entità **DATORE** dotato di un attributo per identificarne la sezione, nell'ottica di futuri sviluppi del progetto. Di conseguenza anche i giorni indisponibili faranno riferimento al datore corrispondente. Ai fini delle nostre specifiche per semplicità supporremo la presenza di un unico datore che faccia riferimento alla sezione Dipendenti e che imposterà i giorni indisponibili per questi ultimi.

Il **modello E-R** completo, cardinalità incluse, è riportato di seguito:



Si ricorda che l'asterisco presente in alcuni attributi indica la possibilità di assumere valori nulli.

Per ogni entità si è scelto un identificatore primario (esterno in alcuni casi) sulla base degli attributi già disponibili, mentre nei casi in cui non fosse possibile la corretta identificazione di una chiave primaria è stato aggiunto un opportuno codice univoco.

In alcuni casi per semplificare la scrittura di alcune funzionalità nel codice sono presenti attributi ridondanti (dati derivati):

- ORE\_DA\_RECUPERARE potrebbe essere ricavato tramite un conteggio di occorrenze delle ore dei turni previsti che non sono diventati turni effettivi.
- Un ragionamento simile permetterebbe di capire quando è il momento di chiudere un servizio (aggiornandone lo STATO\_SERVIZIO), in caso di assenza di personale in numero tale da non poterlo garantire.
- STIPENDIO\_MENSILE potrebbe essere ricavato tramite un conteggio delle occorrenze degli stipendi parziali rispetto all'accredito precedente.

In una fase successiva di sviluppo del progetto sarà possibile valutare in termini di prestazioni se è opportuno mantenere tali ridondanze o meno, mettendo sulla bilancia gli accessi necessari per calcolare i dati derivati, l'occupazione di memoria (oggi spesso trascurabile per attributi di questo tipo) e la necessità di effettuare operazioni aggiuntive per mantenere il dato aggiornato. Per completare tale valutazione sono necessarie ulteriori informazioni sulla specifica azienda che userà il software al momento non disponibili, in particolare il volume dei dati (ad esempio il numero di dipendenti, maggiori dettagli sui servizi e così via) e le caratteristiche delle operazioni (ad esempio il numero medio di esecuzioni in un certo intervallo di tempo).

### 3.4.2 Traduzione Modello E-R

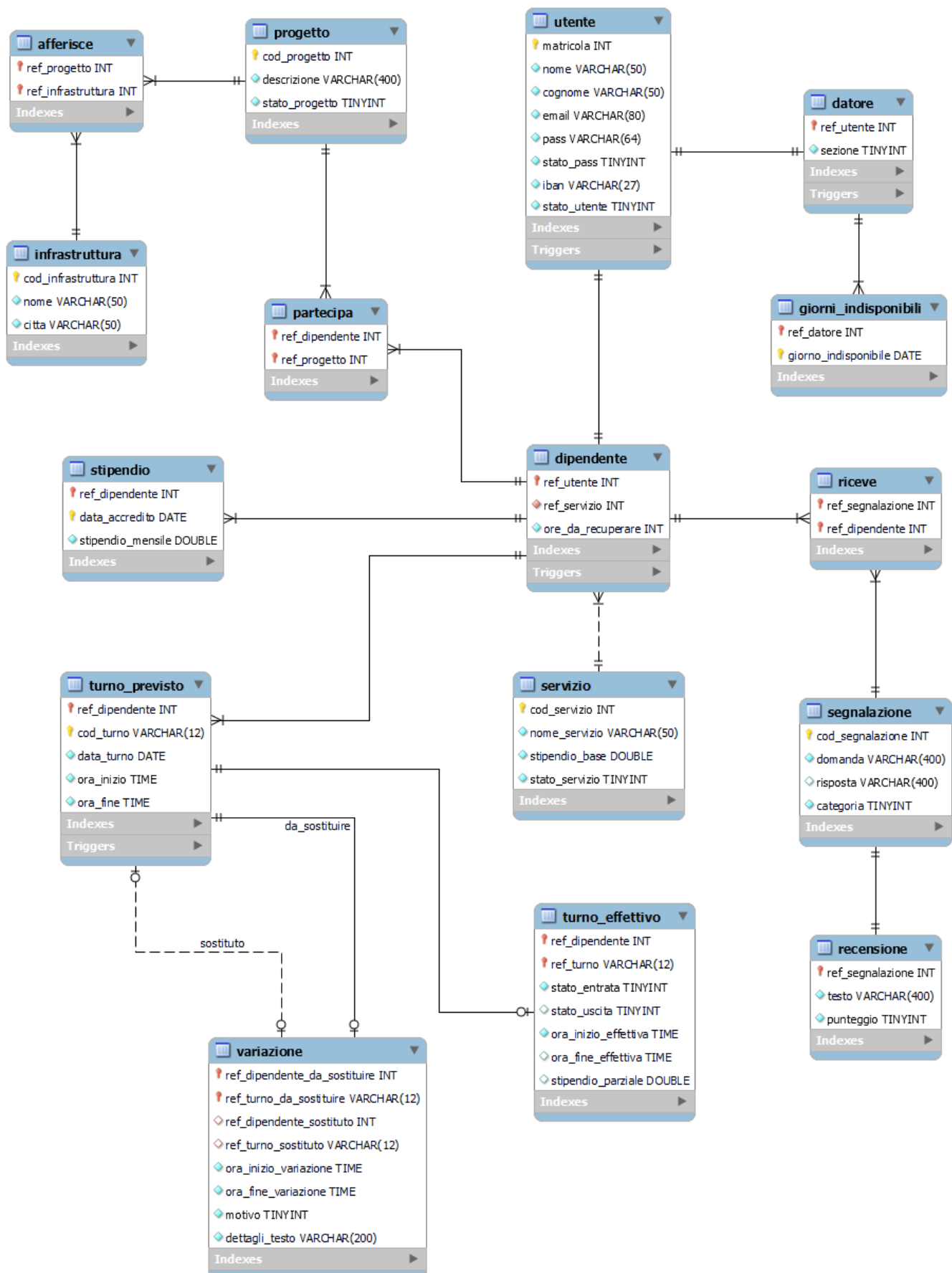
Discutiamo adesso le scelte adottate in fase di traduzione del modello.

Le entità vanno tradotte tutte. Per via delle cardinalità (1,1) non è invece necessario tradurre le seguenti relazioni: SVOLGE, PERCEPISCE, CORRISPONDE, COINCIDE, DA SOSTITUIRE, SOSTITUTO, ASSEGNATO.

Si è scelto di implementare tre tabelle per la generalizzazione totale ed esclusiva: utente (entità genitore), dipendente e datore (entità figlie). La generalizzazione si trasforma in due relazioni uno a uno (che non verranno tradotte) con identificatore esterno che legano rispettivamente l'entità genitore con le entità figlie. Nello schema ottenuto vanno però aggiunti dei vincoli, discussi in seguito. L'alternativa di sostituire la generalizzazione con relazioni converrebbe quando la generalizzazione non è totale, sebbene ciò non sia necessario. Tuttavia essendo presenti nel software funzionalità comuni a tutti gli utenti, funzionalità riservate ai dipendenti e funzionalità riservate al datore, in questo modo è possibile verificare velocemente per ogni funzionalità chi ha diritto a svolgerla, quindi mostrare le schermate corrette e in alcuni casi rendere le query più snelle. È però necessario effettuare operazioni di join per ottenere dati completi per le funzionalità stesse. Altri approcci che prevedevano la traduzione di due tabelle (accorpamento del genitore della generalizzazione nelle figlie) o di un'unica tabella (accorpamento delle figlie della generalizzazione nel genitore), pur non richiedendo join, rendevano il modello più complesso ai fini del progetto.

Viene infine eliminato l'attributo multivalore GIORNI\_INDISPONIBILI e tradotto in un'opportuna tabella.

Il **modello relazionale** ottenuto presenta le seguenti tabelle:

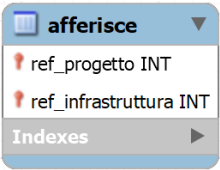
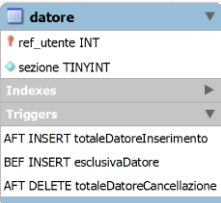
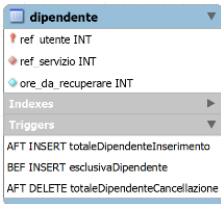


Anche se non serviranno particolari vincoli, occorre ricordare che:

- Non tutti i turni previsti corrispondono a turni effettivi (in caso di variazioni o assenze non comunicate in anticipo).
- In alcuni e rari casi a un turno previsto può corrispondere sia un turno effettivo sia una variazione (in caso di permessi retribuiti chiesti solo su una parte di turno).
- Non tutte le variazioni potrebbero andare a buon fine (in caso di impossibilità a trovare un sostituto).

Queste precisazioni ci permettono di capire perché la linea della relazione `da_sostituire` presente tra `turno_previsto` e `variazione` è continua, mentre quella della relazione `sostituto` presente tra `turno_previsto` e `variazione` è tratteggiata. Si tratta rispettivamente di *identifying* (anche dette *strong*) e *non-identifying* (anche dette *weak*) *relationships*. Nel primo caso la chiave primaria di `variazione` contiene elementi della chiave primaria di `turno_previsto`, ovvero l'esistenza di `variazione` deve dipendere dall'esistenza del turno previsto del dipendente da sostituire (perché le variazioni si baseranno sui turni dei dipendenti stessi). Nel secondo caso invece la chiave primaria di `variazione` non può contenere elementi della chiave primaria di `turno_previsto`, ovvero la `variazione` deve esistere a prescindere dall'esistenza del turno del dipendente sostituto (perché non è detto che si riesca a trovare il sostituto, ma la `variazione` per il corretto funzionamento dell'azienda deve essere memorizzata a prescindere). Un ragionamento analogo si applica alla relazione tra `dipendente` e `servizio`: la chiave primaria di `dipendente` non contiene elementi della chiave primaria di `servizio`.


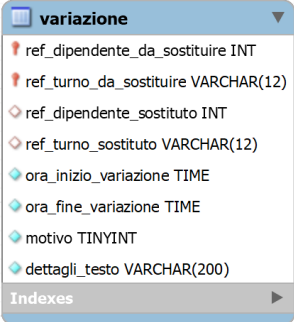
Le singole tabelle, riportate in ordine alfabetico per semplificare la ricerca nel documento, sono di seguito descritte in dettaglio.

Tabella	Descrizione
	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_progetto</code> identifica univocamente il progetto.</li> <li>• L'attributo <code>ref_infrastruttura</code> identifica univocamente l'infrastruttura.</li> </ul>
	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_utente</code> identifica univocamente il datore.</li> <li>• L'attributo <code>sezione</code> identifica la sezione dell'azienda in cui il datore amministra. Può assumere i seguenti valori: 1 (Sezione Dipendenti).</li> </ul>
	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_utente</code> identifica univocamente il dipendente.</li> <li>• L'attributo <code>ref_servizio</code> identifica il servizio in cui il dipendente lavora.</li> <li>• L'attributo <code>ore_da_recuperare</code> memorizza le ore che il dipendente deve recuperare a seguito di assenze e/o variazioni.</li> </ul>

<div>giorni_indisponibili</div> <ul style="list-style-type: none"> <li>ref_datore INT</li> <li>giorno_indisponibile DATE</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>ref_datore</code> identifica univocamente il datore che ha impostato il giorno indisponibile.</li> <li>L'attributo <code>giorno_indisponibile</code> identifica il giorno suddetto.</li> </ul>
<div>infrastruttura</div> <ul style="list-style-type: none"> <li>cod_infrastruttura INT</li> <li>nome VARCHAR(50)</li> <li>citta VARCHAR(50)</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>cod_infrastruttura</code> identifica univocamente l'infrastruttura.</li> <li>Gli attributi <code>nome</code> e <code>città</code> descrivono i rispettivi dati dell'infrastruttura.</li> </ul>
<div>partecipa</div> <ul style="list-style-type: none"> <li>ref_dipendente INT</li> <li>ref_progetto INT</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>ref_dipendente</code> identifica univocamente il dipendente.</li> <li>L'attributo <code>ref_progetto</code> identifica univocamente il progetto.</li> </ul>
<div>progetto</div> <ul style="list-style-type: none"> <li>cod_progetto INT</li> <li>descrizione VARCHAR(400)</li> <li>stato_progetto TINYINT</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>cod_progetto</code> identifica univocamente il progetto.</li> <li>L'attributo <code>descrizione</code> fornisce una descrizione testuale del progetto.</li> <li>L'attributo <code>stato_progetto</code> ne descrive lo stato. Può assumere i seguenti valori: 0 (In pausa), 1 (In corso), 2 (Completato).</li> </ul>
<div>recensione</div> <ul style="list-style-type: none"> <li>ref_segna1azione INT</li> <li>testo VARCHAR(400)</li> <li>punteggio TINYINT</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>ref_segna1azione</code> identifica univocamente la recensione.</li> <li>L'attributo <code>testo</code> contiene il testo della recensione.</li> <li>L'attributo <code>punteggio</code> esprime il grado di soddisfazione della recensione. Può assumere i seguenti valori: 1 (Molto insoddisfatto), 2 (Insoddisfatto), 3 (Né soddisfatto né insoddisfatto), 4 (Soddisfatto), 5 (Molto soddisfatto).</li> </ul>
<div>riceve</div> <ul style="list-style-type: none"> <li>ref_segna1azione INT</li> <li>ref_dipendente INT</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>ref_segna1azione</code> identifica univocamente la segnalazione.</li> <li>L'attributo <code>ref_dipendente</code> identifica univocamente il dipendente.</li> </ul>
<div>segnalazione</div> <ul style="list-style-type: none"> <li>cod_segna1azione INT</li> <li>domanda VARCHAR(400)</li> <li>risposta VARCHAR(400)</li> <li>categoria TINYINT</li> </ul> <div>Indexes</div>	<ul style="list-style-type: none"> <li>L'attributo <code>cod_segna1azione</code> identifica univocamente la segnalazione.</li> <li>Gli attributi <code>domanda</code> e <code>risposta</code> contengono i rispettivi dati della segnalazione.</li> <li>L'attributo <code>categoria</code> identifica la categoria della segnalazione. Può assumere i seguenti valori: 1 (Reclamo), 2 (Proposte e suggerimenti), 3 (Richiesta informazioni), 4 (Altro).</li> </ul>

<div> <div>servizio</div> <div> <div>cod_servizio INT</div> <div>nome_servizio VARCHAR(50)</div> <div>stipendio_base DOUBLE</div> <div>stato_servizio TINYINT</div> </div> <div>Indexes</div> </div>	<ul style="list-style-type: none"> <li>• L'attributo <code>cod_servizio</code> identifica univocamente il servizio. Può assumere i seguenti valori: 1 (Gestione Infrastrutture), 2 (Gestione Progetti), 3 (Gestione Segnalazioni), 4 (Gestione Recensioni).</li> <li>• Gli attributi <code>nome_servizio</code> e <code>stipendio_base</code> descrivono i rispettivi dati del servizio.</li> <li>• L'attributo <code>stato_servizio</code> descrive lo stato del servizio. Può assumere i seguenti valori: 0 (Non si sta svolgendo regolarmente), 1 (Si sta svolgendo regolarmente).</li> </ul>
<div> <div>stipendio</div> <div> <div>ref_dipendente INT</div> <div>data_accredito DATE</div> <div>stipendio_mensile DOUBLE</div> </div> <div>Indexes</div> </div>	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_dipendente</code> identifica univocamente il dipendente.</li> <li>• L'attributo <code>data_accredito</code> identifica la data di accredito dello stipendio. Potrebbero essere necessari alcuni giorni lavorativi affinché l'importo venga effettivamente accreditato.</li> <li>• L'attributo <code>stipendio_mensile</code> identifica l'importo dello stipendio stesso.</li> </ul>
<div> <div>turno_effettivo</div> <div> <div>ref_dipendente INT</div> <div>ref_turno VARCHAR(12)</div> <div>stato_entrata TINYINT</div> <div>stato_uscita TINYINT</div> <div>ora_inizio_effettiva TIME</div> <div>ora_fine_effettiva TIME</div> <div>stipendio_parziale DOUBLE</div> </div> <div>Indexes</div> </div>	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_dipendente</code> identifica univocamente il dipendente previsto in turno.</li> <li>• L'attributo <code>ref_turno</code> identifica univocamente il turno previsto.</li> <li>• L'attributo <code>stato_entrata</code> identifica lo stato del dipendente in entrata. Può assumere i seguenti valori: 0 (Assente), 1 (Presente), 2 (Presente in ritardo).</li> <li>• L'attributo <code>stato_uscita</code> identifica lo stato del dipendente in uscita. Può assumere i seguenti valori: 0 (Chiusura Automatica), 1 (Chiusura Regolare).</li> <li>• Gli attributi <code>ora_inizio_effettiva</code> e <code>ora_fine_effettiva</code> descrivono i rispettivi dati del turno effettivo.</li> <li>• L'attributo <code>stipendio_parziale</code> identifica l'importo dello stipendio riferito allo specifico turno effettivo.</li> </ul>
<div> <div>turno_previsto</div> <div> <div>ref_dipendente INT</div> <div>cod_turno VARCHAR(12)</div> <div>data_turno DATE</div> <div>ora_inizio TIME</div> <div>ora_fine TIME</div> </div> <div>Indexes</div> <div>Triggers</div> <div>BEF INSERT generaCodiceTurno</div> </div>	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_dipendente</code> identifica univocamente il dipendente previsto in turno.</li> <li>• L'attributo <code>cod_turno</code> identifica univocamente il turno previsto. Ulteriori dettagli sono presenti al termine della tabella.</li> <li>• Gli attributi <code>data_turno</code>, <code>ora_inizio</code> e <code>ora_fine</code> descrivono i rispettivi dati del turno previsto.</li> </ul>



	<ul style="list-style-type: none"> <li>• L'attributo <code>matricola</code> identifica univocamente l'utente. Prevediamo che la matricola iniziale sia 1111 e che vengano creati dei vuoti in fase di cancellazione.</li> <li>• Gli attributi <code>nome</code>, <code>cognome</code>, <code>email</code>, <code>pass</code> e <code>iban</code> descrivono i rispettivi dati dell'utente.</li> <li>• L'attributo <code>stato_pass</code> permette di capire lo stato di validità della password. Può assumere i seguenti valori: 0 (È il primo accesso), 1 (Non è il primo accesso). In futuro potrebbe tenere traccia dei giorni mancanti alla scadenza della password.</li> <li>• L'attributo <code>stato_utente</code> permette di capire lo stato dell'utente. Può assumere i seguenti valori: 0 (Inserimento non corretto), 1 (Inserimento corretto), 2 (Lunga astensione dal lavoro). Altri dettagli sull'attributo sono descritti in seguito.</li> </ul>
	<ul style="list-style-type: none"> <li>• L'attributo <code>ref_dipendente_da_sostituire</code> identifica univocamente il dipendente da sostituire.</li> <li>• L'attributo <code>ref_turno_da_sostituire</code> identifica univocamente il turno da sostituire.</li> <li>• L'attributo <code>ref_dipendente_sostituto</code> identifica univocamente il dipendente sostituto.</li> <li>• L'attributo <code>ref_turno_sostituto</code> identifica univocamente il turno sostituto.</li> <li>• Gli attributi <code>ora_inizio_variazione</code> e <code>ora_fine_variazione</code> descrivono i rispettivi dati della variazione.</li> <li>• L'attributo <code>motivo</code> identifica il motivo della variazione. Può assumere i seguenti valori: 1 (Malattia), 2 (Ferie), 3 (Congedo parentale), 4 (Sciopero), 5 (Permesso).</li> <li>• L'attributo <code>dettagli_testo</code> contiene dettagli testuali inseriti dal dipendente in fase di richiesta della variazione.</li> </ul>

Si ricorda che in MySQL il formato del tipo DATE è 'YYYY-MM-DD', mentre quello del tipo TIME è 'hh:mm:ss'.

Poiché per l'azienda tutti i turni iniziano e finiscono a orari con minuti uguali a 00, il formato dei 12 caratteri scelti per il codice turno e relativi riferimenti è 'YYYYMMDDiiff', dove *ii* indica l'orario di inizio turno, mentre *ff* indica l'orario di fine turno. Il codice è generato automaticamente da un opportuno TRIGGER. I quattro attributi `data_turno`, `ora_inizio`, `ora_fine` e `ref_utente` avrebbero permesso a prescindere l'identificazione di una chiave primaria in `turno_previsto`. Tuttavia si è scelto di introdurre il codice turno (che di fatto come detto non è altro che la concatenazione dei tre campi) in modo da ridurre in

turno\_effettivo e soprattutto in variazione il numero di attributi in vincolo di integrità referenziale con turno\_previsto.

Rimanendo in tema, osserviamo che le tabelle tradotte al fine di mantenere coerenza tra i dati inseriti presentano i seguenti **vincoli di integrità referenziale**:

- Tra ref\_utente di dipendente e (la chiave primaria di) utente.
- Tra ref\_servizio di dipendente e (la chiave primaria di) servizio.
- Tra ref\_utente di datore e (la chiave primaria di) utente.
- Tra ref\_datore di giorni\_indisponibili e (la chiave primaria di) datore.
- Tra ref\_dipendente di partecipa e (la chiave primaria di) dipendente.
- Tra ref\_progetto di partecipa e (la chiave primaria di) progetto.
- Tra ref\_progetto di afferisce e (la chiave primaria di) progetto.
- Tra ref\_infrastruttura di afferisce e (la chiave primaria di) infrastruttura.
- Tra ref\_dipendente di riceve e (la chiave primaria di) dipendente.
- Tra ref\_segnalazione di riceve e (la chiave primaria di) segnalazione.
- Tra ref\_segnalazione di recensione e (la chiave primaria di) segnalazione.
- Tra ref\_dipendente di stipendio e (la chiave primaria di) dipendente.
- Tra ref\_dipendente di turno\_previsto e (la chiave primaria di) dipendente.
- Tra ref\_dipendente e ref\_turno di turno\_effettivo e (la chiave primaria di) turno\_previsto.
- Tra ref\_dipendente\_da\_sostituire e ref\_turno\_da\_sostituire di variazione e (la chiave primaria di) turno\_previsto.
- Tra ref\_dipendente\_sostituto e ref\_turno\_sostituto di variazione e (la chiave primaria di) turno\_previsto.

Per una maggiore robustezza e per snellire eventuali controlli sui dati in fase di scrittura del codice, sono stati usati in maniera opportuna i **vincoli presenti in SQL**, come UNSIGNED (per garantire che alcuni attributi siano positivi), UNIQUE (per garantire che alcuni attributi non presentino duplicati) e NOT NULL (per garantire che alcuni attributi in certe situazioni non presentino valori nulli, evitando ambiguità o incoerenze). In alcuni casi in fase di inserimento si useranno valori nulli per attributi non noti a priori (ad esempio in fase di rilevazione entrata e conseguente inserimento in turno\_effettivo non sono ancora noti i dettagli dell'uscita). Abbiamo preferito questa soluzione rispetto all'utilizzo di un valore sentinella (ovvero un valore scelto in modo tale che non possa essere confuso con un valore accettabile).

La base di dati gestisce inoltre **vincoli non esprimibili in maniera diretta**, di seguito descritti.

Tramite vincoli CHECK si garantisce che:

- Il formato dell'IBAN sia corretto (facendo riferimento alle coordinate bancarie italiane, ma il vincolo è facilmente adattabile a qualsiasi stato).
- Il formato di alcuni attributi che possono assumere solo valori prestabiliti (descritti nella tabella del modello relazione) sia corretto.

- Gli stipendi siano positivi (essendo l'uso di unsigned deprecato per valori in virgola mobile).
- Un dipendente non possa sostituire se stesso.

I **vincoli di generalizzazione** per utente, dipendente e datore sono così gestiti:

- L'esclusività ( $dipendente \cap datore = \emptyset$ ) è garantita da una coppia di TRIGGER che impone il vincolo di disgiunzione tra dipendente e datore: in fase di inserimento di una tupla (identificabile univocamente da un attributo in vincolo con la matricola) in una delle due sottoclassi, viene verificato che nell'altra sottoclasse non esista una tupla il cui attributo in vincolo con la matricola presenti lo stesso valore.
- La totalità ( $dipendente \cup datore = utente$ ) non è invece immediata da garantire tramite trigger, perché non è semplice stabilire a quale tabella dovrebbe fare riferimento un ipotetico trigger che dovrebbe imporre il vincolo di unione. Il problema della totalità riguarda l'inserimento di tuple in utente che non abbiano una tupla corrispondente in una delle due sottoclassi (ad esempio a seguito di un errore o di una caduta di connessione). Mettiamoci in questa situazione critica e analizziamo i due scenari possibili. Se il trigger viene attivato in seguito all'inserimento di una tupla in utente, non abbiamo molto altro da controllare perché siamo certi che in dipendente/datore non sia stata inserita la tupla corrispondente. Verrebbe quindi spontaneo attivare il trigger in seguito all'inserimento di una tupla in una delle due sottoclassi, ma abbiamo detto che nella situazione critica questo inserimento non è avvenuto (e anche perché se fosse avvenuto, data la presenza di un vincolo di integrità referenziale, sicuramente in utente era già presente la tupla corrispondente, eliminando quindi ogni dubbio). Arrivati a questo punto, oltre all'utilizzo delle transazioni (ed eventuale rollback, ovvero riportare la base di dati a una versione precedente in caso qualcosa vada storto) che sicuramente è buona norma, riteniamo opportuno aggiungere in utente un attributo `stato_utente`. L'attributo diventerebbe 1 solo in seguito a un inserimento completo e privo di incoerenze: un opportuno TRIGGER a seguito dell'inserimento in dipendente o in datore modificherebbe il valore di `stato_utente` da 0 a 1 (e non ha neanche bisogno di verificare la presenza della tupla corrispondente su utente, dato che c'è un vincolo di integrità referenziale può direttamente cambiare lo stato senza ulteriori accertamenti). Il controllo del valore di `stato_utente` permetterebbe di garantire la totalità in modo rapido. Terminata la fase di inserimento l'attributo non rimarrebbe sprecato: potrebbe essere utile in una futura versione per identificare utenti (dipendenti soprattutto) che per un certo intervallo di tempo non svolgeranno regolarmente il loro lavoro (per lunghe malattie o per congedo parentale ad esempio) e che non dovranno quindi essere coinvolti dal sistema nella generazione dei turni o delle sostituzioni. A questo punto però entra in gioco un secondo TRIGGER, che deve garantire che dopo la cancellazione da una sottoclasse venga cancellata anche la tupla corrispondente in utente. È una situazione improbabile perché la cancellazione dovrebbe avvenire sempre a partire da utente, ma riteniamo utile gestire questo scenario in modo da assicurare sempre la consistenza dei dati.

Al momento abbiamo scelto una politica di cancellazione a cascata in caso di licenziamento di un dipendente, anche perché è stato supposto nei requisiti non funzionali che assunzioni e licenziamenti avvengano sempre prima della generazione dei turni e del calcolo degli stipendi, così da non intaccare le funzionalità automatiche. Abbiamo scelto come descritto poco sopra di garantire che vengano cancellati

anche i dati corrispondenti nella tabella utente (incluse le credenziali di accesso) in modo da garantire la totalità della generalizzazione. In futuro, in base alle specifiche politiche dell'azienda e a particolari vincoli legali, potrebbe essere aggiunto un secondo database in cui archiviare tutti i dati dei dipendenti e dei datori passati e decidere in questo scenario di mantenere alcuni profili attivi per accedere a funzionalità riservate. L'identificazione di questi profili potrebbe avvenire sempre attraverso l'attributo `stato_utente`, che assumerebbe un ulteriore valore (3).

### 3.5 Access control and security

Nel Requirements Analysis Document abbiamo associato casi d'uso diversi ad attori diversi, modellando così le distinzioni di accesso alle singole funzionalità. In futuro sarà possibile modellare in maniera più formale tali distinzioni attraverso una *access matrix*, rappresentandola attraverso uno dei tre approcci tra *global access table*, *access control list*, and *capabilities*.

Sono inoltre impostati vincoli sulle funzionalità per impedirne l'uso improprio qualora necessario (ad esempio la funzionalità Gestione Servizi non è disponibile al di fuori del proprio orario di lavoro, così come i singoli servizi sono accessibili solo dai dipendenti ad essi assegnati).

Per quanto riguarda i requisiti di sicurezza, come già descritto in alcuni casi d'uso presenti nel Requirements Analysis Document, sono previste due modalità di accesso al software. In particolare per alcune operazioni come la rilevazione dell'entrata e dell'uscita dei dipendenti è sufficiente inserire nel terminale dell'azienda nome, cognome e matricola. Per le operazioni che possono coinvolgere dati personali più sensibili come permettere ai dipendenti di consultare la propria situazione lavorativa/stipendiale o permettere al datore di avere una panoramica generale dell'azienda è invece necessario inserire in fase di login matricola e password.

All'interno del database è rischioso memorizzare le password di tutti gli utenti *in chiaro*, soprattutto nel caso in cui venga violato. Pertanto si è preferito memorizzare i 64 caratteri esadecimali ottenuti applicando a ogni singola password la funzione crittografica di hash SHA-256.

Inoltre a livello implementativo verrà assicurata la prevenzione di eventuali *SQL injection* mediante l'utilizzo di query parametrizzate.

In fase di assunzione di un nuovo dipendente viene generata una password temporanea che dovrà obbligatoriamente essere modificata al primo accesso: essendo stata inviata in chiaro per email preferiamo che venga cambiata quanto prima e come detto un apposito attributo permetterà di tenere sotto controllo questa situazione.

Tutti i login, incluso il primo, sono protetti da una verifica in due passaggi, che prevede a seguito dell'inserimento della password corretta anche l'inserimento di un codice temporaneo inviato all'utente per email. Abbiamo ritenuto utile implementare questa funzionalità poiché molti utenti tendono purtroppo a impostare password facili da indovinare o peggio a scriverle su fogli di carta o file.

In futuro potranno essere aggiunti vincoli specifici dettati dalla specifica azienda che userà il software, ad esempio aggiungere una scadenza per le password, imporre un particolare formato (lunghezza minima, presenza di caratteri maiuscoli e minuscoli, presenza di numeri, presenza di caratteri speciali), oppure

imporre di usare password diverse dalle precedenti. A seconda del tipo di dispositivi presenti in azienda potranno anche essere usati sensori biometrici.

### 3.6 Global software control

In questa prima versione del System Design Document non è previsto lo sviluppo di questo paragrafo.

### 3.7 Boundary conditions

Per gestire alcune situazioni impreviste che potrebbero verificarsi interagendo con il DBMS, il metodo che eseguirà le query provvederà in caso di caduta di connessione a riproporre l'interrogazione effettuando 3 tentativi di riconnessione.

In futuro potranno essere aggiunti comportamenti più specifici per tutte le situazioni impreviste, nonché descrizioni che riguardano l'avvio e l'arresto del sistema. Come già detto in precedenza, il terminale per la rilevazione della presenza, il pc aziendale e il repository rimarranno sempre attivi 24 ore su 24. Di conseguenza per alcuni nodi avvio e arresto rappresentano situazioni eccezionali dovute a problemi di natura elettrica, elettronica e informatica.

## 4. Subsystem services

In questa prima versione del System Design Document non è previsto lo sviluppo di questo paragrafo.

## 5. Glossary

Le definizioni del glossario (al momento solo in lingua inglese) sono *riusate* dal libro *Object-Oriented Software Engineering: Using Uml, Patterns and Java* di Bernd Bruegge e Allen H. Dutoit. Abbiamo selezionato quelle che fanno principalmente riferimento al SDD. Altri termini utili possono essere trovati nel RAD e nell'ODD.

A	
Architectural style ( <i>Architettura</i> )	A general system design model that can be used as a starting point for system design. Examples of architectural styles include client/server, peer to peer, pipe and filter, and model/view/controller.
C	
Client/server architectural style ( <i>Architettura client-server</i> )	An architectural style in which user interactions are managed by simple client programs and functionality is delivered by a central server program.
Component ( <i>Componente</i> )	A physical and replaceable part of the system that complies to an interface. Examples of components include class libraries, frameworks, and binary programs
D	
Deployment diagram (...)	A UML diagram representing run-time components and their assignments to hardware nodes.

Design goal (...)	A quality that the system should optimize. Design goals are often inferred from nonfunctional requirements and are used to guide design decisions. Examples of design goals include usability, reliability, security, and safety.
<b>L</b>	
Layer (...)	A subsystem in a hierarchical decomposition. A layer can depend only on lower-level layers and has no knowledge of the layers above it.
<b>M</b>	
Mapping (...)	A mathematical correspondence that assigns an element (or a set of elements) of one model to each element of another model.
Model/View/Controller (MVC) architectural style ( <i>Architettura MVC</i> )	A three-tier architectural style in which domain knowledge is maintained by model objects, displayed by view objects, and manipulated by control objects. In this book, model objects are called entity objects and view objects are called boundary objects.
<b>P</b>	
Peer-to-peer architectural style ( <i>Architettura peer-to-peer</i> )	A generalization of the client/server architectural style in which subsystems can act either as clients or servers.
Persistent data ( <i>Dati persistenti</i> )	Data that outlive a single execution of the system.
Pipe and filter architectural style ( <i>Architettura pipe and filter</i> )	An architectural style in which subsystems sequentially process data from a set of inputs and send their results to other subsystems via a set of outputs. Associations between subsystems are called pipes. Subsystems are called filters. Filters do not depend on each other and can thus be rearranged in different orders and configurations.
Primary key ( <i>Chiave primaria</i> )	In a database management system, a set of attributes whose values uniquely identify the data records in a table.
<b>R</b>	
Repository architectural style ( <i>Architettura repository</i> )	An architectural style in which persistent data is managed and stored by a single subsystem. Peripheral subsystems are relatively independent and interact only through the central subsystem.
<b>S</b>	
Service ( <i>Servizio</i> )	A set of related operations offered by a subsystem.
Subsystem ( <i>Sottosistema</i> )	In general, a smaller, simpler part of a larger system; in system design, a welldefined software component that provides a number of services to other subsystems. Examples of subsystems include storage subsystems (managing persistent data), user interface subsystems (managing the interaction with the user), networking

	subsystems (managing the communication with other subsystems over a network).
Subsystem decomposition ( <i>Decomposizione in sottosistemi</i> )	The division of the system into subsystems. Each subsystem is described in terms of its services during system design and its API during object design. Subsystem decomposition is part of the system design model.
System ( <i>Sistema</i> )	An organized set of communicating parts designed for a specific purpose. For example, a car, composed of four wheels, a chassis, a body, and an engine, is designed to transport people. A watch, composed of a battery, a circuit, wheels, and hands, is designed to measure time.
System Design Document (SDD)	A document describing the system design model.
System design model (...)	A high-level description of the system, including design goals, subsystem decomposition, hardware/software platform, persistent storage strategy, global control flow, access control policy, and boundary condition strategies. The system design model represents the strategic decisions made by the architecture team that allow subsystem teams to work concurrently and cooperate effectively.
<b>T</b>	
Three-tier architectural style ( <i>Architettura three-tier</i> )	An architectural style that organizes the system into an interface layer, an application logic layer, and a storage layer.
Tuple ( <i>Tupla</i> )	An ordered set of values. Common uses for the tuple are representing a set of value attributes in a relational database and representing an access right.