Sviluppo delle Applicazioni Software

Laboratorio

a.a 2023/24

Vito Fasulo 986492
Matteo Di Noia 979620

Gestire i turni

Informazioni generali

Nome caso d'uso: Gestire i turni
Portata: Sistema
Livello: Obiettivo Utente
Attore primario: Organizzatore
Parti Interessate: Personale
Pre-condizioni: L'attore deve essere autenticato come Organizzatore
Post-condizioni: L'inserimento/modifica dei turni sono visualizzabili dalle parti interessate

Scenario principale di successo

#	Attore	Sistema
1.	Organizzatore apre Evento esistente	Fornisce dati evento
2.	Apre orario di cucina o di servizio	Fornisce orario di cucina o servizio
3.	Opzionalmente ottiene lista dei turni	Fornisce lista dei turni
	Prosegue a 4 oppure termina	
4.	Crea e inserisce dati turno (data, ora, durata, num. persone richieste, ricorrenza, data chiusura disiscrizione, etc)	Registra nuovo turno
	Ripete passi da 3 finché non soddisfatto	

Estensioni

Eccezione 1a

#	Attore	Sistema
1a.1	Organizzatore apre Evento esistente	Evento non appartiene all'organizzatore.
	Termina il caso d'uso	

Eccezione 4a

#	Attore	Sistema
4a.1	Crea e inserisce dati turno (data, ora, durata, num. persone richieste, ricorrenza, data chiusura disiscrizione, etc)	Il turno creato è sovrapposto, anche parzialmente, ad un altro turno (in stesso luogo).

Eccezione 4b

#	Attore	Sistema
4b.1	Crea e inserisce dati turno (data, ora, durata, num. persone richieste, ricorrenza, data chiusura disiscrizione, etc)	Il turno creato ha dati non sensati (ora negativa, persone negativa, data chiusura minore di data, data prima di giornata attuale).

Estensione 4c

#	Attore	Sistema
4c.1	aggrega/disaggrega i turni	Registra aggregazione

Estensione 4d

#	Attore	Sistema
4d.1	Modifica i dati (data, ora,durata, num. persone richieste, ricorrenza) di uno o più turni	Registra modifica dei dati

Eccezione 4d.1a

#	Attore	Sistema
4d.1a.1	Modifica i dati (data, ora,durata, persone richieste, ricorrenza) di uno o più turni	Il turno modificato è ora sovrapposto, anche parzialmente, ad un altro turno (in stesso luogo).

Eccezione 4d.1b

#	Attore	Sistema
4d.1b.1	Modifica i dati (data, ora,durata, persone richieste, ricorrenza) di uno o più turni	Il turno modificato ha dati non sensati (ora negativa, persone negativa, data chiusura minore di data, data prima di giornata attuale).

Estensione 4e

#	Attore	Sistema
4e.1	sblocca forzatamente o togli sblocco forzato su gruppo turni	Registra modifica dei dati

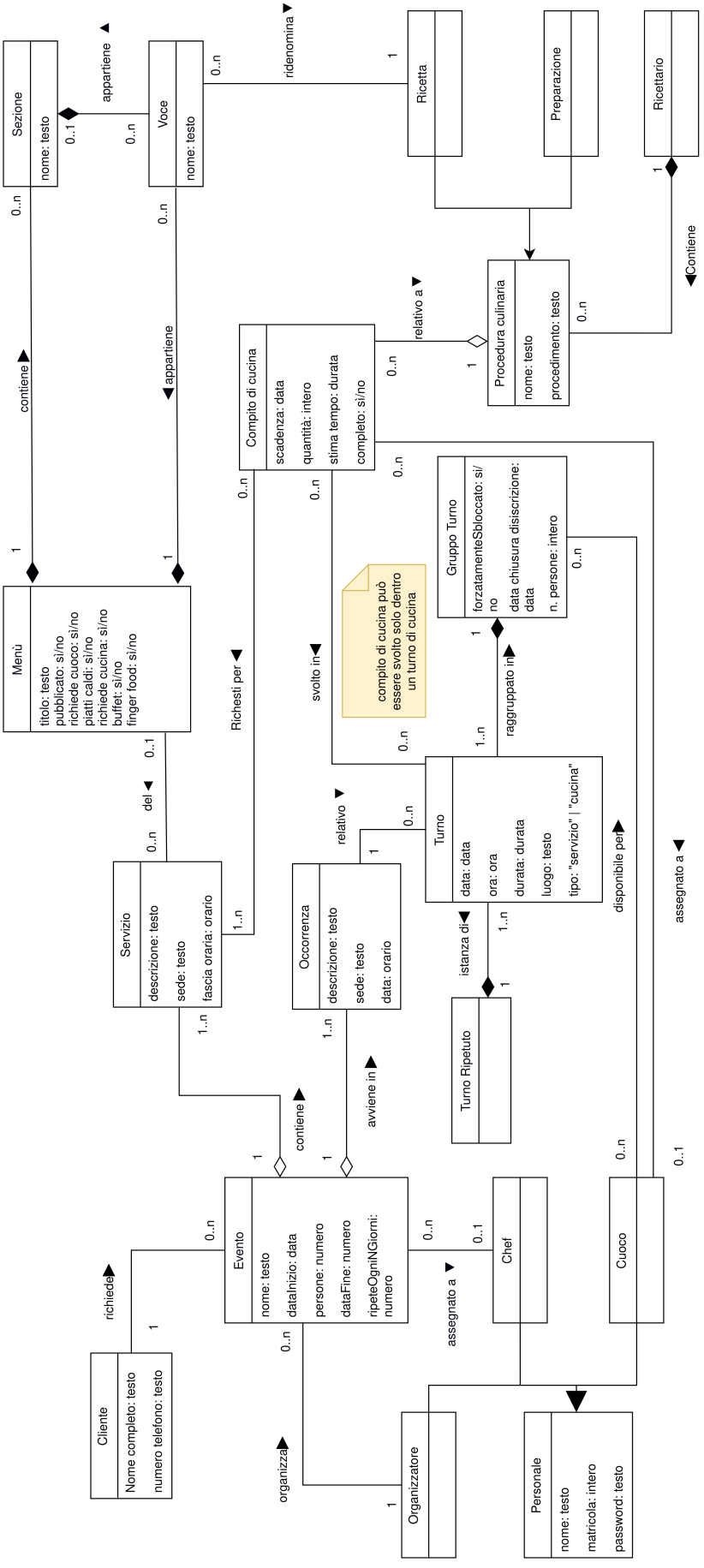
Eccezione 4e.1a

#	Attore	Sistema
4e.1a.1	sblocca forzatamente o togli sblocco forzato su gruppo turni	Il gruppo di turni è compreso di soli turni scaduti.

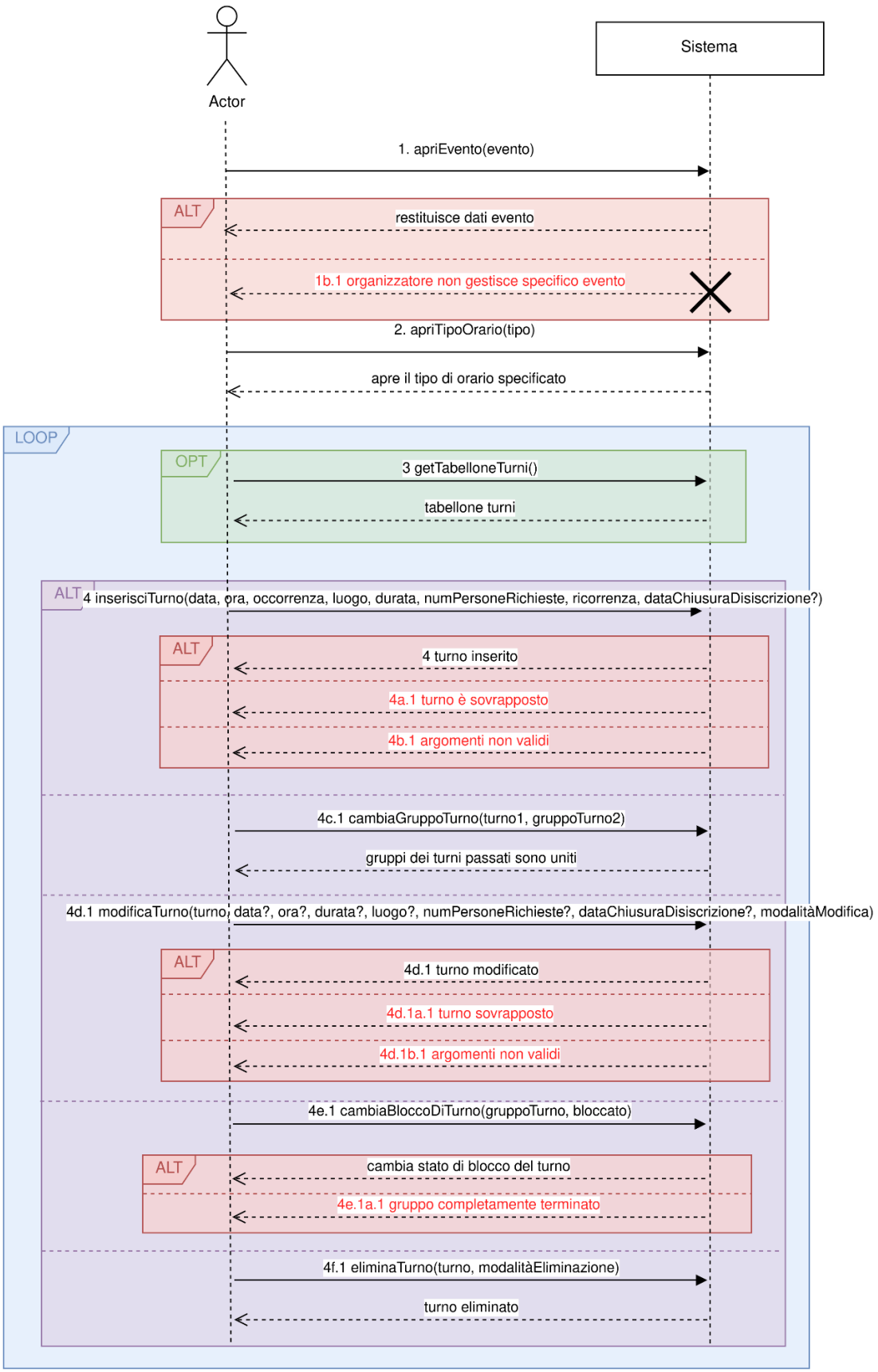
Estensione 4f

#	Attore	Sistema
4f.1	elimina turno/i selezionato/i	Registra eliminazione

Modello di Dominio



SSD


Actor
Sistema
1. apriEvento(evento)
ALT
restituisce dati evento
1b.1 organizzatore non gestisce specifico evento
2. apriTipoOrario(tipo)
apre il tipo di orario specificato
LOOP
OPT
3 getTabelloneTurni()
tabellone turni
ALT
4 inserisciTurno(data, ora, occorrenza, luogo, durata, numPersoneRichieste, ricorrenza, dataChiusuraDisiscrizione?)
ALT
4 turno inserito
4a.1 turno è sovrapposto
4b.1 argomenti non validi
4c.1 cambiaGruppoTurno(turno1, gruppoTurno2)
gruppi dei turni passati sono uniti
4d.1 modificaTurno(turno, data?, ora?, durata?, luogo?, numPersoneRichieste?, dataChiusuraDisiscrizione?, modalitàModifica)
ALT
4d.1 turno modificato
4d.1a.1 turno sovrapposto
4d.1b.1 argomenti non validi
4e.1 cambiaBloccoDiTurno(gruppoTurno, bloccato)
ALT
cambia stato di blocco del turno
4e.1a.1 gruppo completamente terminato
4f.1 eliminaTurno(turno, modalitàEliminazione)
turno eliminato

Contratti

Pre-condizione generale: L'utente deve essere autentificato come Organizzatore.

1-ApriEvento(evento: Evento)

Pre-condizioni:
- L'evento esiste ed è **organizzato** dall'organizzatore correntemente autenticato.

Post-condizioni: _

2-apriTipoOrario(tipo: "servizio" | "cucina")

Pre-condizioni: _
Post-condizioni: _

3-getTabelloneTurni()

Pre-condizioni: _
Post-condizioni: _

4-inserisciTurno(data: Data, ora: Ora, occorrenza: Occorrenza, luogo: Testo, durata: Durata, numPersoneRichieste: numero, ricorrenza: si/no, dataChiusuraDisiscrizione?: Data)

Pre-condizioni:
- data e ora rappresentano un orario futuro
- dataChiusuraDisiscrizione è posteriore a data
- E' aperto l'Evento evento ed è aperto il Tipo tipo
- Il turno non esiste un altro turno *t2* tale che t1!=t2 con *t2*.tipo=turno.tipo e *t2*.luogo=turno.luogo

Post-condizioni:
- E' stata creata istanza *turno* di Turno
- *turno* è **relativo** a occorrenza
- [se ricorrenza è impostata] E' stata creata istanza TurnoRipetuto *turnoRipetuto*
    - *turno* è **istanza di** *turnoRipetuto*
    - Per ogni istanza *occorrenzaSucc* di Occorrenza che **avviene in** evento, ed è posteriore ad occorrenza, è stato creato turno *turnoSucc*
        - *turnoSucc* ***è relativo*** *ad occorrenzaSucc*
        - *turnoSucc* è **istanza di** *turnoRipetuto*
        - È stato creato *gruppoTurnoSucc* istanza di GruppoTurno
        - *turnoSucc* è raggruppato in *gruppoTurnoSucc*
- È stato creato *gruppoTurno* istanza di GruppoTurno
- *turno* è raggruppato in *gruppoTurno*

4c.1-cambiaGruppoTurno(turno1: Turno, gruppoTurno2?: GruppoTurno)

Pre-condizioni:

- turno1 è raggruppato in *gruppoTurno1* istanza di GruppoTurno

Post-condizioni:

- [Se gruppoTurno2 è definito]
  - Per ogni *cuoco* istanza di Cuoco disponibile per gruppoTurno2 o *gruppoTurno1*, cuoco **non è più disponibile** per gruppoTurno2 o *gruppoTurno1*
  - Per ogni *t* istanza di Turno **raggruppato in** *gruppoTurno1*, allora t **non è più raggruppato** a *gruppoTurno1* ma **è raggruppato** a gruppoTurno2
  - L'istanza *gruppoTurno1* è stato eliminata
- [Se gruppoTurno2 non è definito e *gruppoTurno1* **raggruppa** più di un Turno] E' stato istanziato *nuovoGruppo* di tipo GruppoTurno
  - turno1 **non è più raggruppato** in *gruppoTurno1* ma **è raggruppato** in *nuovoGruppo*

4-modificaTurno(turno: Turno, data?: Data, ora?: Ora, luogo?: Testo, durata?: Durata, numPersoneRichieste?: numero, dataChiusuraDisiscrizione?: Data, modalitàModifica: "singolo" | "successivi" | "tutti")

Pre-condizioni:

- turno esiste
- [Se esiste data] *diffData* è la diffenza tra la data e turno.data
- [Se non esiste data] *diffData = 0*
- [Se esiste dataChiusuraDisiscrizione] *diffDataChius* è la diffenza tra la dataChiusuraDisiscrizione e turno.dataChiusuraDisiscrizione
- [Se non esiste data] *diffDataChius = 0*

Post-condizioni:

- [Se modalitàModifica è singolo o *turno* **non è istanza** di un TurnoRipetuto] *turniInteressati* contiene solo turno
- [Se modalitàModifica non è singolo] dato *turnoRipetuto* di cui turno **è istanza,** *turniInteressati* contiene tutti i turni che **sono istanza** di *turnoRipetuto* successivi o uguali a turno se modalitàModifica="successivi" o tutti altrimenti
- Per ogni *turno* contenuto di *turniInteressati*
  - turno.data = turno.data + *diffData*
  - turno.dataChiusuraDisiscrizione = turno.dataChiusuraDisiscrizione + diffDataChius
  - [Se durata esiste] turno.durata = durata
  - [Se ora esiste] turno.ora = ora
  - [Se numPersoneRichiesta esiste] turno.numPersoneRichieste = numPersoneRichieste

4e.1-cambiaStatoSbloccoForzato(gruppoTurno: GruppoTurno, sbloccato: sì/no)

Pre-condizioni:

- Esiste Turno *turno* **raggruppato in** gruppoTurno, tale per cui *turno*.data e *turno*.ora è posteriore all'attuale data e ora

Post-condizioni: _

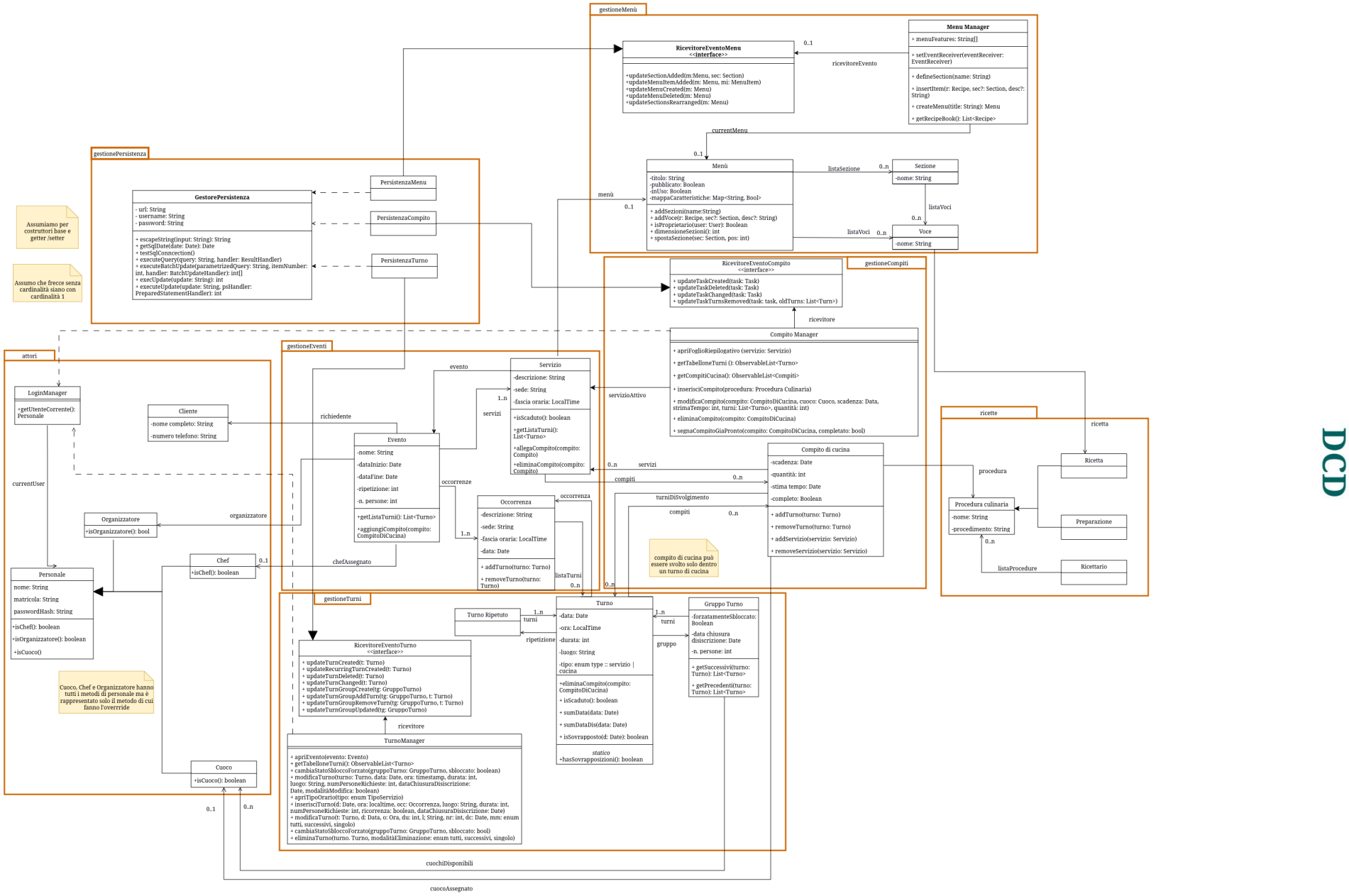
- GruppoTurno.forzatamenteSbloccato = forzatamenteSbloccato

4f.1-eliminaTurno(turno: Turno, modalitàEliminazione: “singolo” | “successivi” | “tutti“)

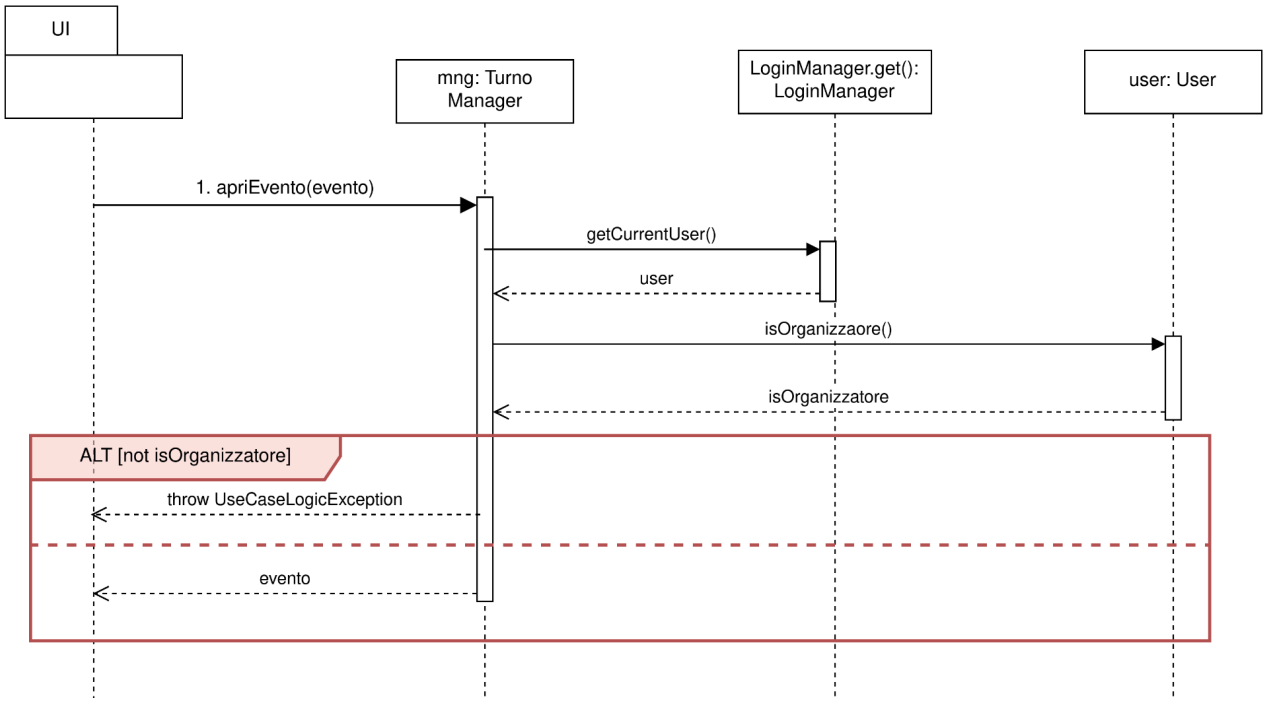
Pre-condizioni: _

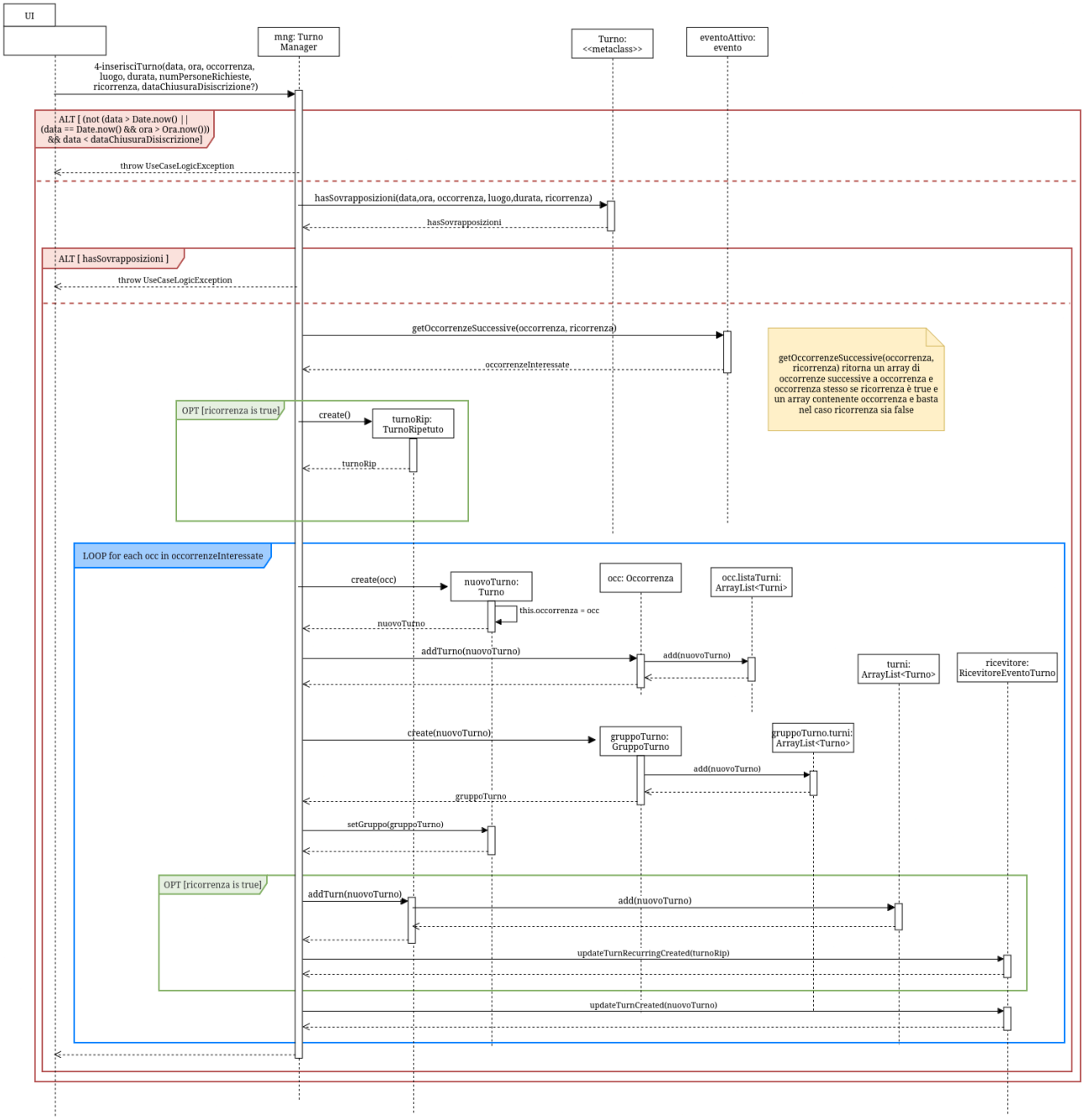
Post-condizioni:

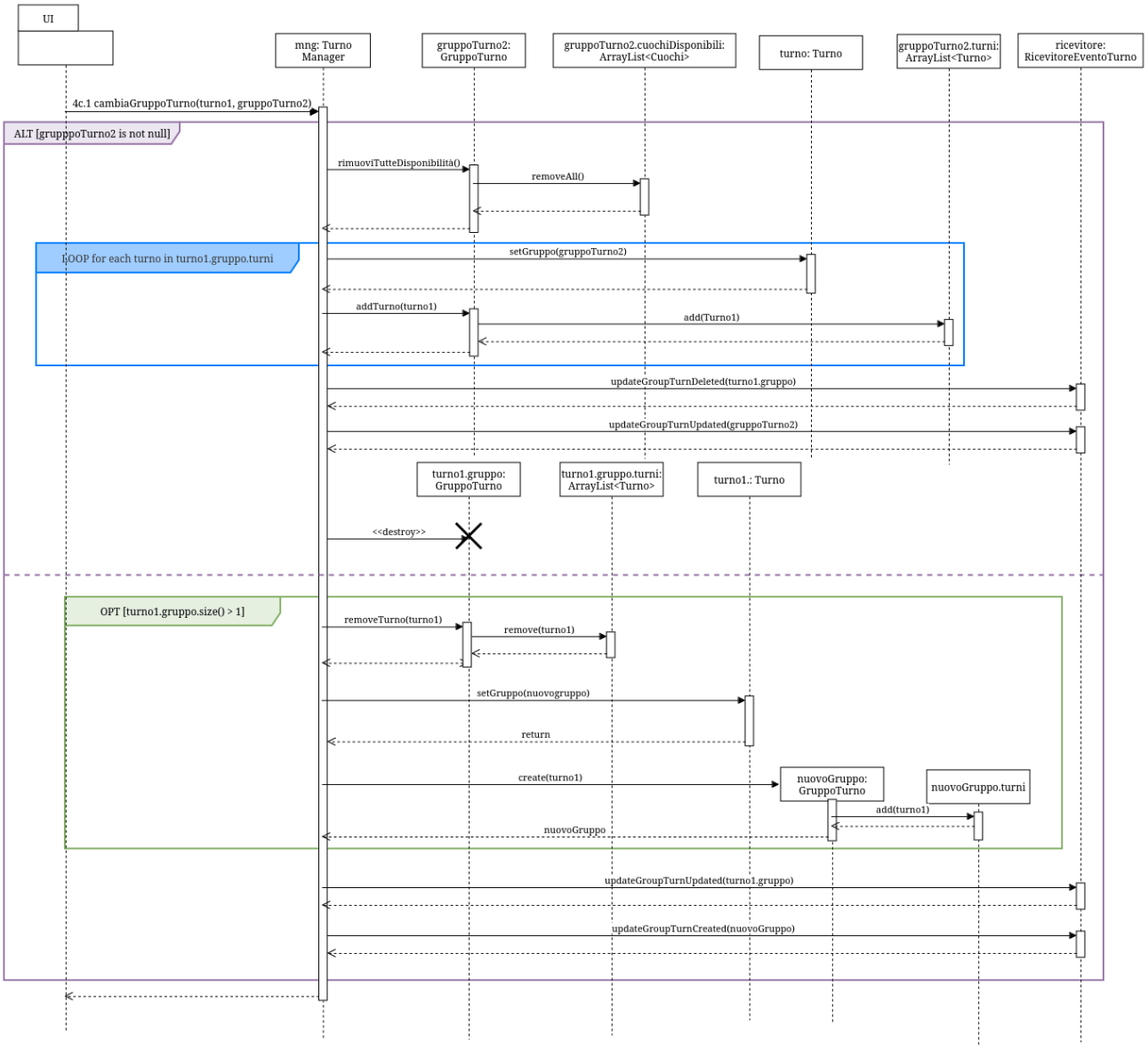
- [Se non esiste TurnoRipetuto di cui **è istanza** turno oppure modalitàModifica="singolo"] *turniInteressati* contiene solo turno
- [Se il turno **è istanza di** TurnoRipetuto *turnoRipetuto* e modalitàModifica!="singolo"] *turniInteressati* contiene ogni Turno **istanza** di *turnoRipetuto*, se modalitàModifica=”successivi” allora vengono considerati solo quelli successivi a turno; tutti altrimenti
- Per ogni *turnoAltro* in *turniInteressati*
    - Data *turnoAltro* che è **relativo all’**Occorrenza *occorrenza*, *turnoAltro* **non è più relativo a** *occorrenza*
    - Per ogni CompitiDiCucina compito il quale **è svolto** in *turnoAltro*, *turnoAltro* **non svolge più** *compito*
    - Dato *GruppoTurno* gruppo in cui turno **è raggruppato**, *turnoAltro* **non è più raggruppato** in *gruppo*
        - [Se non c’è Turno che **è raggruppato in** *gruppo*] per ogni Cuoco *cuoco* **disponibile per** *gruppo*, *cuoco* **non è più disponibile** per *gruppo* ed è stata eliminata l’istanza *gruppo.*
    - [Se non c’è Turno che **è istanza di** *turnoRipetuto*] è stata eliminata l’istanza *turnoRipetuto*
    - E’ stata eliminata l’istanza *turnoAltro*

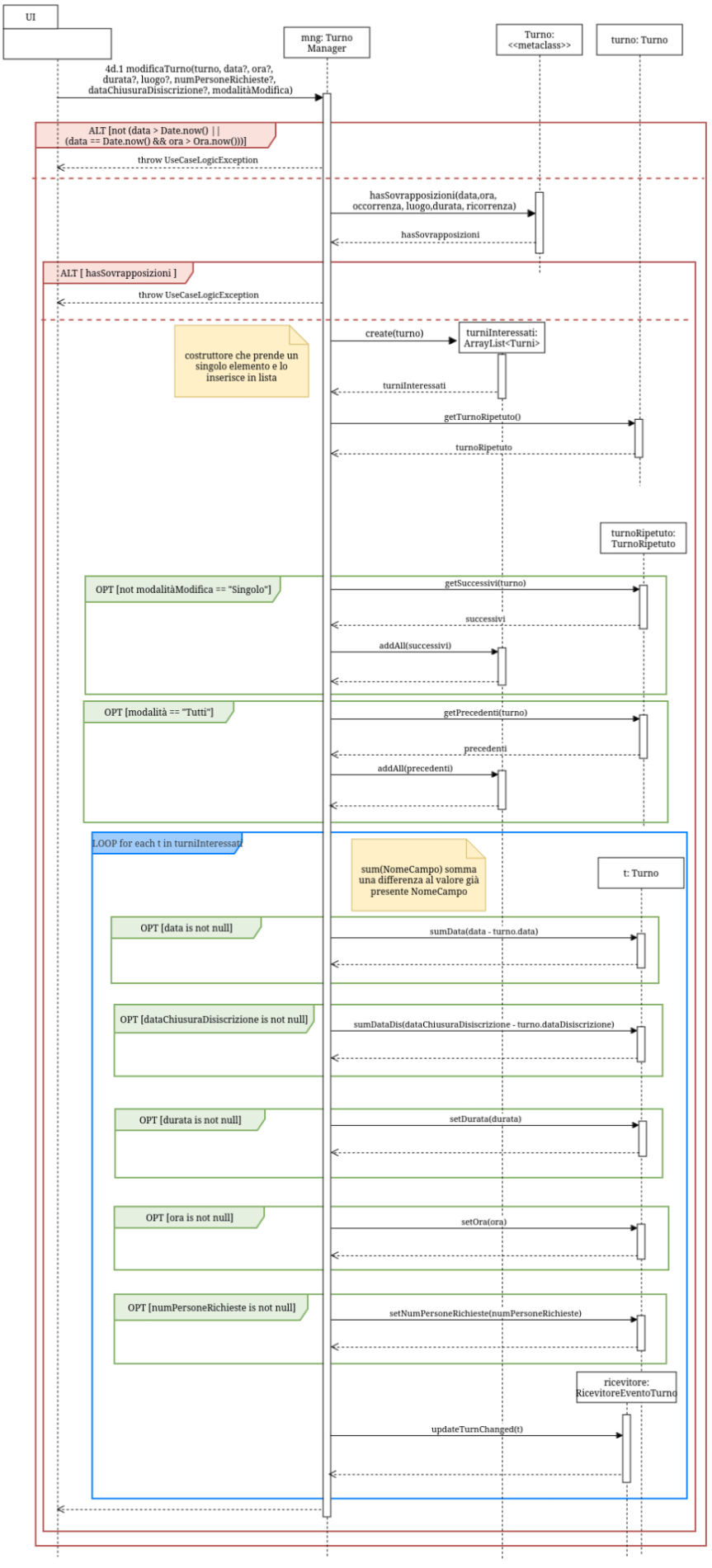

gestioneMenù
RicevitoreEventoMenu
<<interface>>
+updateSectionAdded(m:Menu, sec: Section)
+updateMenuItemAdded(m: Menu, mi: MenuItem)
+updateMenuCreated(m: Menu)
+updateMenuDeleted(m: Menu)
+updateSectionsRearranged(m: Menu)
0..1
ricevitoreEvento
Menu Manager
+ menuFeatures: String[]
+ setEventReceiver(eventReceiver: EventReceiver)
+ defineSection(name: String)
+ insertItem(r: Recipe, sec?: Section, desc?: String)
+ createMenu(title: String): Menu
+ getRecipeBook(): List<Recipe>
currentMenu
0..1
Menù
-titolo: String
-pubblicato: Boolean
-intUso: Boolean
-mappaCaratteristiche: Map<String, Bool>
+ addSezioni(name:String)
+ addVoce(r: Recipe, sec?: Section, desc?: String)
+ isProprietario(user: User): Boolean
+ dimensioneSezioni(): int
+ spostaSezione(sec: Section, pos: int)
listaSezione
0..n
Sezione
-nome: String
listaVoci
0..n
menù
0..1
listaVoci
0..n
Voce
-nome: String
gestionePersistenza
GestorePersistenza
- url: String
- username: String
- password: String
+ escapeString(input: String): String
+ getSqlDate(date: Date): Date
+ testSqlConnection()
+ executeQuery(query: String, handler: ResultHandler)
+ executeBatchUpdate(parametrizedQuery: String, itemNumber: int, handler: BatchUpdateHandler): int[]
+ execUpdate(update: String): int
+ executeUpdate(update: String, psHandler: PreparedStatementHandler): int
PersistenzaMenu
PersistenzaCompito
PersistenzaTurno
Assumiamo per costruttori base e getter /setter
Assumo che frecce senza cardinalità siano con cardinalità 1
gestioneCompiti
RicevitoreEventoCompito
<<interface>>
+ updateTaskCreated(task: Task)
+ updateTaskDeleted(task: Task)
+ updateTaskChanged(task: Task)
+ updateTaskTurnsRemoved(task: task, oldTurns: List<Turn>)
ricevitore
Compito Manager
+ apriFoglioRiepilogativo (servizio: Servizio)
+ getTabelloneTurni (): ObservableList<Turno>
+ getCompitiCucina(): ObservableList<Compiti>
+ inserisciCompito(procedura: Procedura Culinaria)
+ modificaCompito(compito: CompitoDiCucina, cuoco: Cuoco, scadenza: Data, strimaTempo: int, turni: List<Turno>, quantità: int)
+ eliminaCompito(compito: CompitoDiCucina)
+ segnaCompitoGiaPronto(compito: CompitoDiCucina, completato: bool)
attori
LoginManager
+getUtenteCorrente(): Personale
currentUser
Cliente
-nome completo: String
-numero telefono: String
richiedente
Organizzatore
+isOrganizzatore(): bool
organizzatore
Chef
+isChef(): boolean
0..1
chefAssegnato
Personale
nome: String
matricola: String
passwordHash: String
+isChef(): boolean
+isOrganizzatore(): boolean
+isCuoco()
Cuoco, Chef e Organizzatore hanno tutti i metodi di personale ma è rappresentato solo il metodo di cui fanno l'overrride
Cuoco
+isCuoco(): boolean
0..1
0..n
gestioneEventi
evento
Servizio
-descrizione: String
-sede: String
-fascia oraria: LocalTime
+isScaduto(): boolean
+getListaTurni(): List<Turno>
+allegaCompito(compito: Compito)
+eliminaCompito(compito: Compito)
1..n
servizi
servizioAttivo
Evento
-nome: String
-dataInizio: Date
-dataFine: Date
-ripetizione: int
-n. persone: int
+getListaTurni(): List<Turno>
+aggiungiCompito(compito: CompitoDiCucina)
occorrenze
1..n
Occorrenza
-descrizione: String
-sede: String
-fascia oraria: LocalTime
-data: Date
+ addTurno(turno: Turno)
+ removeTurno(turno: Turno)
occorrenza
listaTurni
0..n
0..n
servizi
compiti
0..n
Compito di cucina
-scadenza: Date
-quantità: int
-stima tempo: Date
-completo: Boolean
+ addTurno(turno: Turno)
+ removeTurno(turno: Turno)
+ addServizio(servizio: Servizio)
+ removeServizio(servizio: Servizio)
turniDiSvolgimento
compiti
0..n
compito di cucina può essere svolto solo dentro un turno di cucina
ricette
ricetta
Ricetta
procedura
Procedura culinaria
-nome: String
-procedimento: String
Preparazione
0..n
listaProcedure
Ricettario
gestioneTurni
Turno Ripetuto
1..n
turni
ripetizione
Turno
-data: Date
-ora: LocalTime
-durata: int
-luogo: String
-tipo: enum type :: servizio | cucina
+eliminaCompito(compito: CompitoDiCucina)
+ isScaduto(): boolean
+ sumData(data: Date)
+ sumDataDis(data: Date)
+ isSovrapposto(d: Date): boolean
statico
+hasSovrapposizioni(): boolean
1..n
turni
gruppo
Gruppo Turno
-forzatamenteSbloccato: Boolean
-data chiusura disiscrizione: Date
-n. persone: int
+ getSuccessivi(turno: Turno): List<Turno>
+ getPrecedenti(turno: Turno): List<Turno>
RicevitoreEventoTurno
<<interface>>
+ updateTurnCreated(t: Turno)
+ updateRecurringTurnCreated(t: Turno)
+ updateTurnDeleted(t: Turno)
+ updateTurnChanged(t: Turno)
+ updateTurnGroupCreated(g: GruppoTurno)
+ updateTurnGroupAddTurn(tg: GruppoTurno, t: Turno)
+ updateTurnGroupRemoveTurn(tg: GruppoTurno, t: Turno)
+ updateTurnGroupUpdated(tg: GruppoTurno)
ricevitore
TurnoManager
+ apriEvento(evento: Evento)
+ getTabelloneTurni(): ObservableList<Turno>
+ cambiaStatoSbloccoForzato(gruppoTurno: GruppoTurno, sbloccato: boolean)
+ modificaTurno(turno: Turno, data: Date, ora: timestamp, durata: int, luogo: String, numPersoneRichieste: int, dataChiusuraDisiscrizione: Date, modalitàModifica: boolean)
+ apriTipoOrtartoTipo: enum TipoServizio)
+ inserisciTurno(d: Date, ora: localtime, occ: Occorrenza, luogo: String, durata: int, numPersoneRichieste: int, ricorrenza: boolean, dataChiusuraDisiscrizione: Date)
+ modificaTurno(t: Turno, d: Data, o: Ora, du: int, l: String, nr: int, dc: Date, mm: enum tutti, successivi, singolo)
+ cambiaStatoSbloccoForzato(gruppoTurno: GruppoTurno, sbloccato: bool)
+ eliminaTurno(turno. Turno, modalitàEliminazione: enum tutti, successivi, singolo)
cuochiDisponibili
cuocoAssegnato
DCD

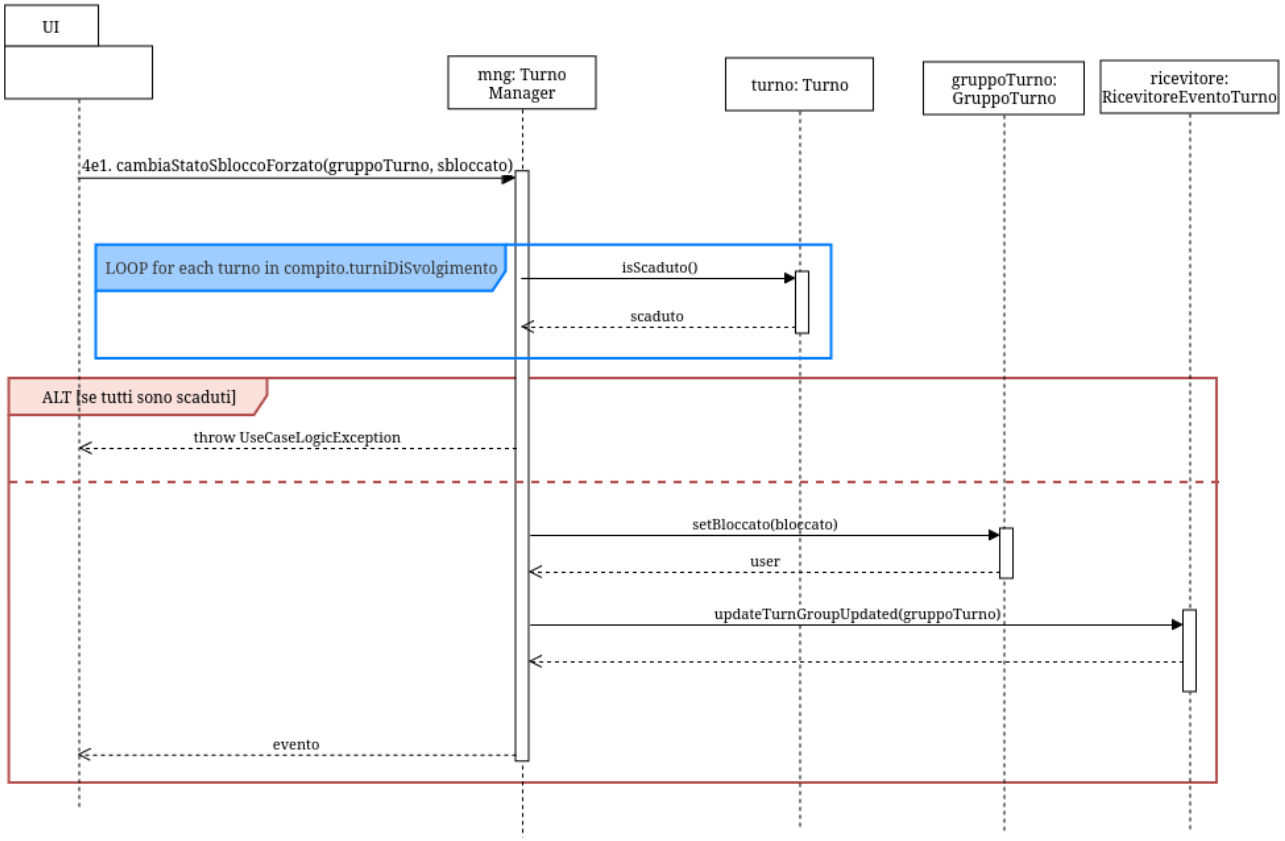
DSD

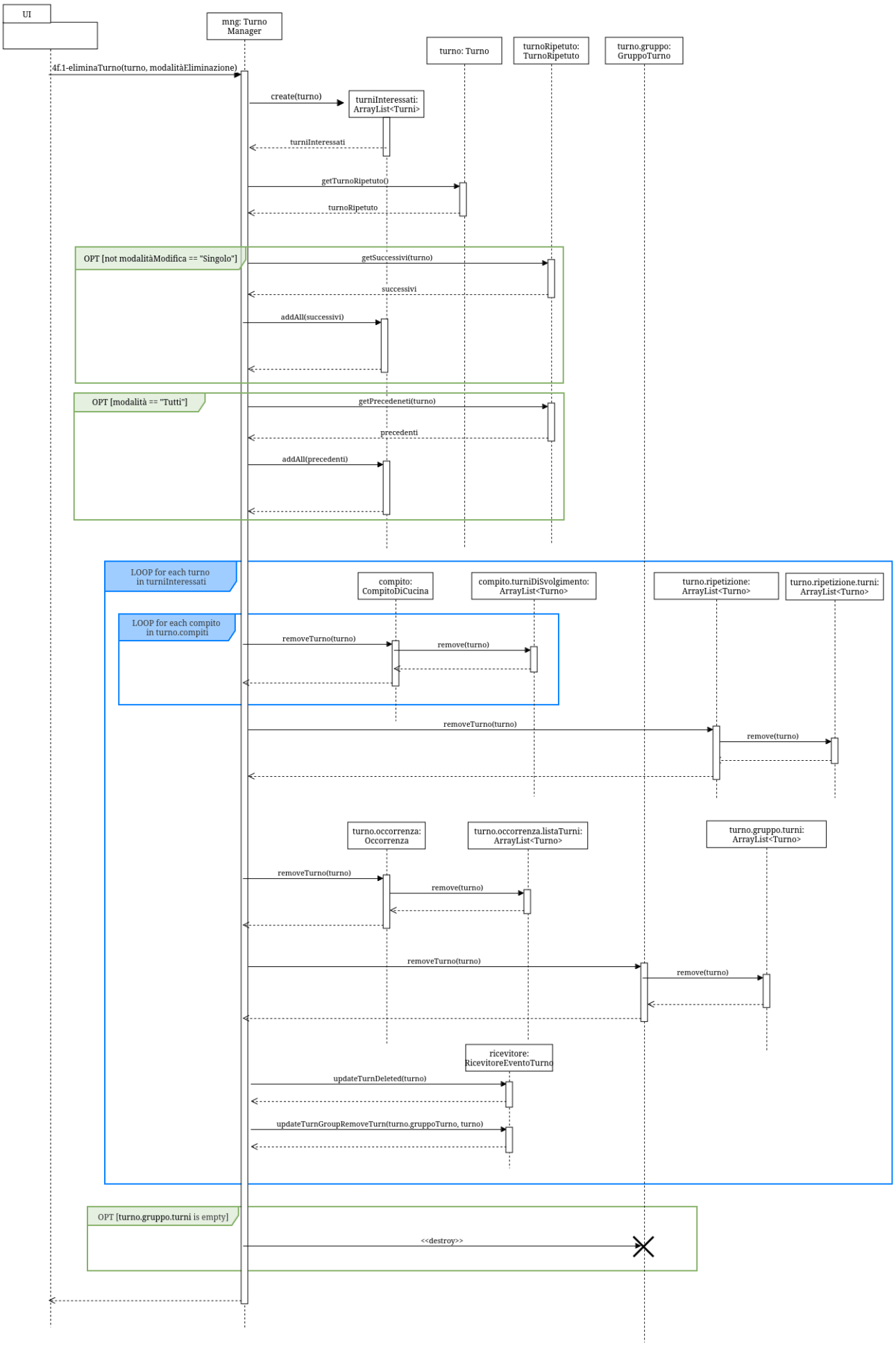

UI
mng: Turno Manager
LoginManager.get(): LoginManager
user: User
1. apriEvento(evento)
getCurrentUser()
user
isOrganizzaore()
isOrganizzatore
ALT [not isOrganizzatore]
throw UseCaseLogicException
evento


UI
mng: Turno Manager
Turno: <<metaclass>>
eventoAttivo: evento
4-inserisciTurno(data, ora, occorrenza, luogo, durata, numPersoneRichieste, ricorrenza, dataChiusuraDisiscrizione?)
ALT [(not (data > Date.now() || (data == Date.now() && ora > Ora.now())) && data < dataChiusuraDisiscrizione]
throw UseCaseLogicException
hasSovrapposizioni(data,ora, occorrenza, luogo,durata, ricorrenza)
hasSovrapposizioni
ALT [hasSovrapposizioni]
throw UseCaseLogicException
getOccorrenzeSuccessive(occorrenza, ricorrenza)
occorrenzeInteressate
getOccorrenzeSuccessive(occorrenza, ricorrenza) ritorna un array di occorrenze successive a occorrenza e occorrenza stesso se ricorrenza è true e un array contenente occorrenza e basta nel caso ricorrenza sia false
OPT [ricorrenza is true]
create()
turnoRip: TurnoRipetuto
turnoRip
LOOP for each occ in occorrenzeInteressate
create(occ)
nuovoTurno: Turno
occ: Occorrenza
occ.listaTurni: ArrayList<Turni>
this.occorrenza = occ
nuovoTurno
addTurno(nuovoTurno)
add(nuovoTurno)
turni: ArrayList<Turno>
ricevitore: RicevitoreEventoTurno
create(nuovoTurno)
gruppoTurno: GruppoTurno
gruppoTurno.turni: ArrayList<Turno>
add(nuovoTurno)
gruppoTurno
setGruppo(gruppoTurno)
OPT [ricorrenza is true]
addTurn(nuovoTurno)
add(nuovoTurno)
updateTurnRecurringCreated(turnoRip)
updateTurnCreated(nuovoTurno)


UI
mng: Turno Manager
gruppoTurno2: GruppoTurno
gruppoTurno2.cuochiDisponibili: ArrayList<Cuochi>
turno: Turno
gruppoTurno2.turni: ArrayList<Turno>
ricevitore: RicevitoreEventoTurno
4c.1 cambiaGruppoTurno(turno1, gruppoTurno2)
ALT [grupppoTurno2 is not null]
rimuoviTutteDisponibilità()
removeAll()
LOOP for each turno in turno1.gruppo.turni
setGruppo(gruppoTurno2)
addTurno(turno1)
add(Turno1)
updateGroupTurnDeleted(turno1.gruppo)
updateGroupTurnUpdated(gruppoTurno2)
turno1.gruppo: GruppoTurno
turno1.gruppo.turni: ArrayList<Turno>
turno1.: Turno
<<destroy>>
OPT [turno1.gruppo.size() > 1]
removeTurno(turno1)
remove(turno1)
setGruppo(nuovogruppo)
return
create(turno1)
nuovoGruppo: GruppoTurno
nuovoGruppo.turni
add(turno1)
nuovoGruppo
updateGroupTurnUpdated(turno1.gruppo)
updateGroupTurnCreated(nuovoGruppo)


UI
mng: Turno Manager
Turno: <<metaclass>>
turno: Turno
4d.1 modificaTurno(turno, data?, ora?, durata?, luogo?, numPersoneRichieste?, dataChiusuraDisiscrizione?, modalitàModifica)
ALT [not (data > Date.now() || (data == Date.now() && ora > Ora.now()))]
throw UseCaseLogicException
hasSovrapposizioni(data,ora, occorrenza, luogo,durata, ricorrenza)
hasSovrapposizioni
ALT [hasSovrapposizioni]
throw UseCaseLogicException
costruttore che prende un singolo elemento e lo inserisce in lista
create(turno)
turniInteressati: ArrayList<Turni>
turniInteressati
getTurnoRipetuto()
turnoRipetuto
turnoRipetuto: TurnoRipetuto
OPT [not modalitàModifica == "Singolo"]
getSuccessivi(turno)
successivi
addAll(successivi)
OPT [modalità == "Tutti"]
getPrecedenti(turno)
precedenti
addAll(precedenti)
LOOP for each t in turniInteressati
sum(NomeCampo) somma una differenza al valore già presente NomeCampo
t: Turno
OPT [data is not null]
sumData(data - turno.data)
OPT [dataChiusuraDisiscrizione is not null]
sumDataDis(dataChiusuraDisiscrizione - turno.dataDisiscrizione)
OPT [durata is not null]
setDurata(durata)
OPT [ora is not null]
setOra(ora)
OPT [numPersoneRichieste is not null]
setNumPersoneRichieste(numPersoneRichieste)
ricevitore: RicevitoreEventoTurno
updateTurnChanged(t)


UI
mng: Turno Manager
turno: Turno
gruppoTurno: GruppoTurno
ricevitore: RicevitoreEventoTurno
4e1. cambiaStatoSbloccoForzato(gruppoTurno, sbloccato)
LOOP for each turno in compito.turniDiSvolgimento
isScaduto()
scaduto
ALT [se tutti sono scaduti]
throw UseCaseLogicException
setBloccato(bloccato)
user
updateTurnGroupUpdated(gruppoTurno)
evento


UI
mng: Turno Manager
turno: Turno
turnoRipetuto: TurnoRipetuto
turno.gruppo: GruppoTurno
4f.1-eliminaTurno(turno, modalitàEliminazione)
create(turno)
turniInteressati: ArrayList<Turni>
turniInteressati
getTurnoRipetuto()
turnoRipetuto
OPT [not modalitàModifica == "Singolo"]
getSuccessivi(turno)
successivi
addAll(successivi)
OPT [modalità == "Tutti"]
getPrecedeneti(turno)
precedenti
addAll(precedenti)
LOOP for each turno in turniInteressati
compito: CompitoDiCucina
compito.turniDiSvolgimento: ArrayList<Turno>
turno.ripetizione: ArrayList<Turno>
turno.ripetizione.turni: ArrayList<Turno>
LOOP for each compito in turno.compiti
removeTurno(turno)
remove(turno)
removeTurno(turno)
remove(turno)
turno.occorrenza: Occorrenza
turno.occorrenza.listaTurni: ArrayList<Turno>
turno.gruppo.turni: ArrayList<Turno>
removeTurno(turno)
remove(turno)
removeTurno(turno)
remove(turno)
ricevitore: RicevitoreEventoTurno
updateTurnDeleted(turno)
updateTurnGroupRemoveTurn(turno.gruppoTurno, turno)
OPT [turno.gruppo.turni is empty]
<<destroy>>