Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

Eco AI

Titolo del documento:

Documento di Architettura

Document Info

Doc. Name	*D3-ECO-AI_Architettura*	Doc. Number	D3
Description	Il documento include diagrammi delle classi e codice in OCL		

INDICE

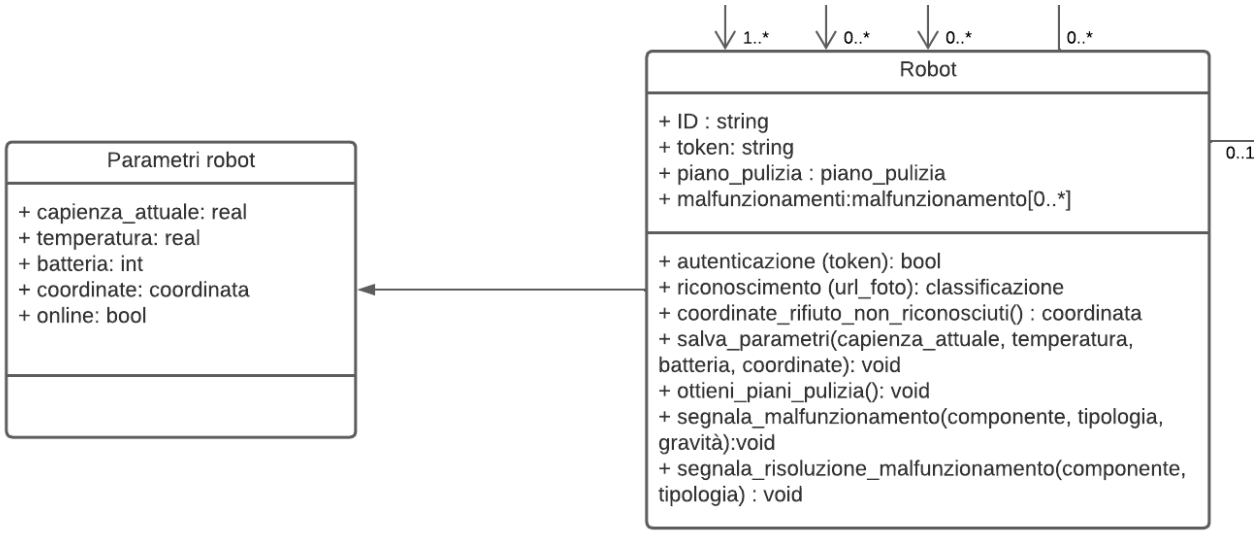
Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto ECO AI usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema dettagliando da un lato le classi che dovranno essere implementate a livello di codice e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML. La logica viene descritta in OCL perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

1. Diagramma delle classi

Analizzando il diagramma di contesto è stato possibile individuare 8 attori: robot, utente, utente anonimo, utente autenticato, utente membro, amministratore secondario, amministratore primario e amministratore del sito.

1.1 Robot

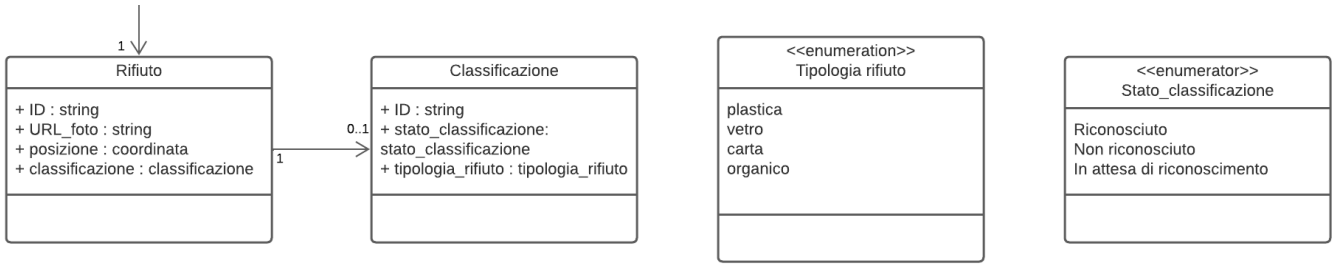


Il robot è l'attore che si occupa del raccoglimento e smistamento dei rifiuti. È stata quindi individuata una classe "Robot" per la gestione delle interazioni che quest'ultimo esegue con il sistema.
Ogni robot è identificato da un ID univoco e possiede un token di autenticazione per interagire con il sistema.
Del robot vengono inoltre memorizzate informazioni quali capienza attuale, temperatura, stato della batteria e posizione attuale. Il robot è in grado di aggiornare il proprio stato ("online" o "offline") attraverso un parametro booleano. Per questo motivo è stata individuata una classe "Parametri robot" che contiene tali informazioni.
Infine, del robot vengono memorizzati i malfunzionamenti riscontrati in una lista.

Il robot si autentica attraverso l'utilizzo di un token univoco. Una volta autenticato, il robot può interagire con il sistema.

Riconoscimento

Il robot richiede un riconoscimento, inviando una foto del rifiuto da riconoscere e la posizione del rifiuto.

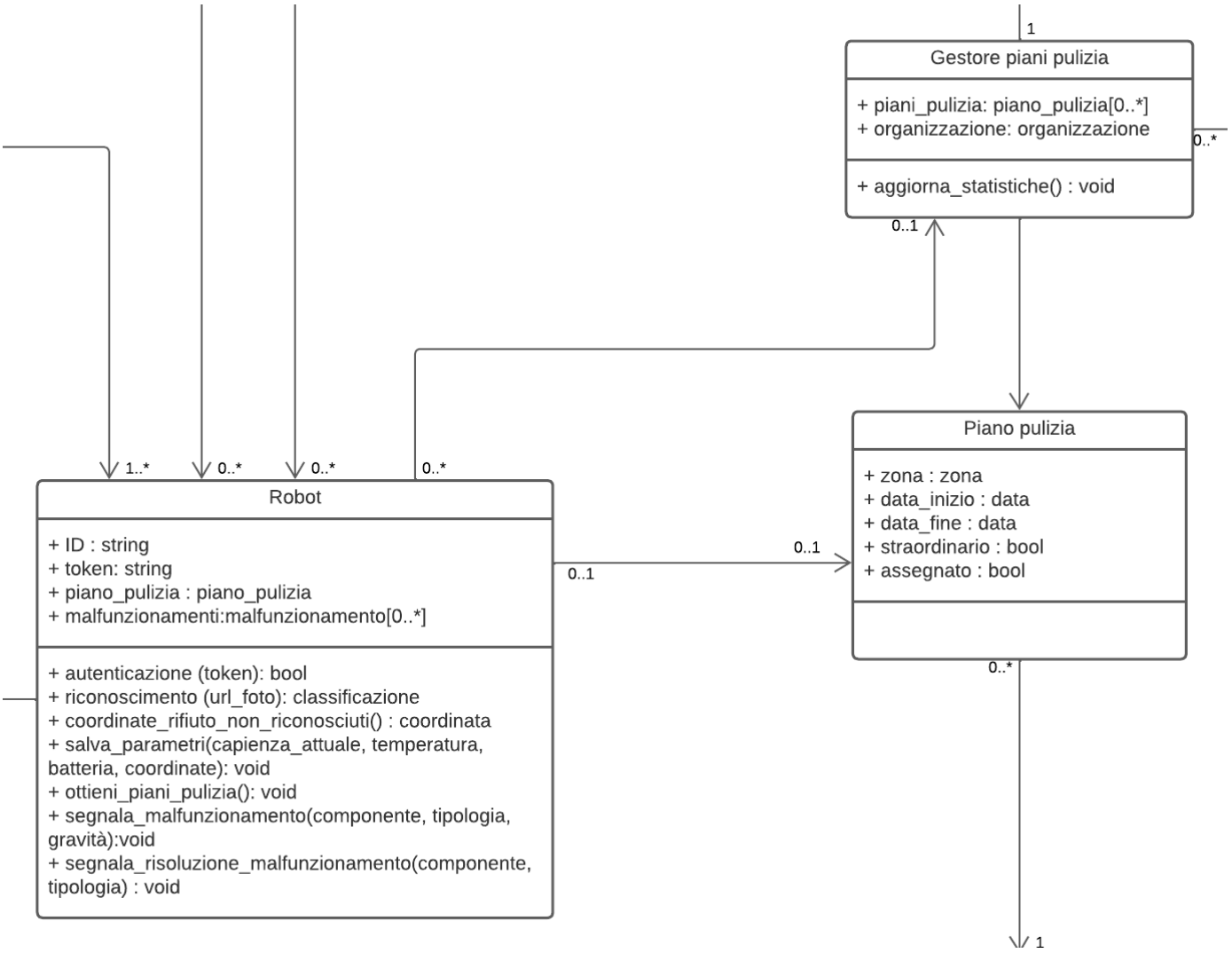


È stata identificata una classe "Classificazione", contenente le informazioni riguardo il tipo di rifiuto. Le tipologie di rifiuto identificate possono essere: plastica, vetro, carta e organico. Si può tenere traccia dello stato di classificazione attraverso l'apposito attributo.
Nel caso in cui un rifiuto non venga riconosciuto, quest'ultimo viene memorizzato nel sistema con stato di classificazione "In attesa di riconoscimento", insieme alla posizione del rifiuto. In seguito viene inviata una notifica ai membri dell'organizzazione di cui il robot fa parte. Il robot procederà ad ignorare il rifiuto, in attesa che esso venga riconosciuto manualmente da un membro dell'organizzazione.

Memorizzazione parametri

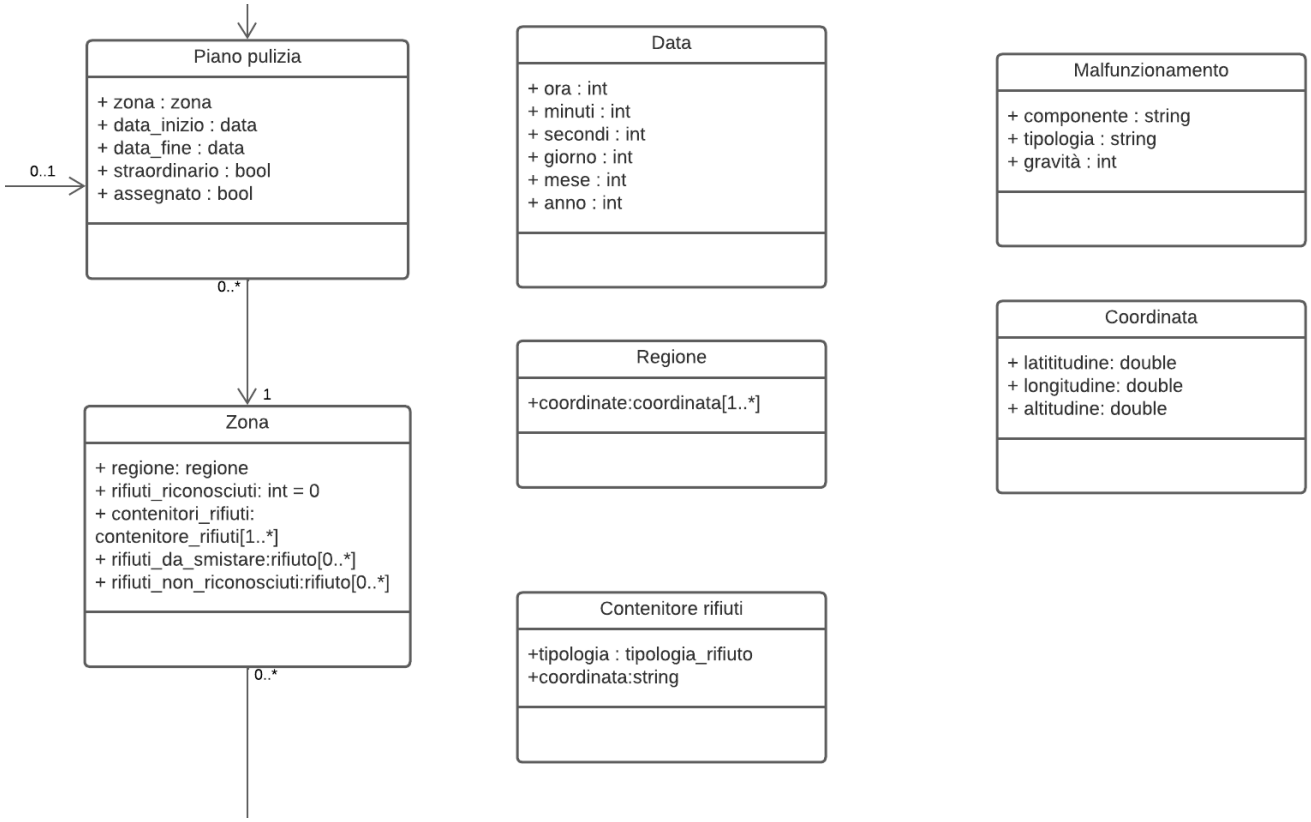
Il robot invia un aggiornamento dei propri parametri (capienza attuale, temperatura, batteria, coordinate, stato).

Piani di pulizia

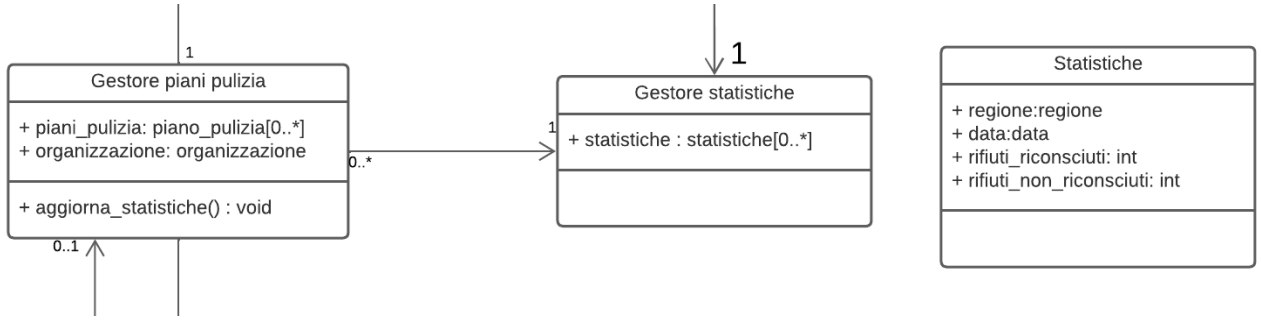


Attraverso il metodo “ottieni_piani_pulizia” il robot richiede al sistema i piani di pulizia. Per fare ciò, sono state identificate 2 classi “Piano pulizia” e “Gestore piani pulizia”. I piani di pulizia vengono gestiti e contenuti dal Gestore piani pulizia, il quale è legato ad una particolare organizzazione. I piani di pulizia sono memorizzati in una coda a priorità, il cui ordine di priorità sarà basato sull’orario di esecuzione (le date più vecchie hanno priorità maggiore). Indipendentemente dall’orario, i piani segnati come “straordinario” avranno la priorità massima.

Il piano di pulizia ha lo scopo di contenere la zona e l'orario in cui il robot andrà ad operare seguendo tale piano. Un piano di pulizia può essere indicato come “straordinario” nel caso debba essere eseguito il prima possibile, dunque riceverà priorità rispetto ai piani ordinari. Quando un piano viene assegnato ad un robot, viene impostata la flag “assegnato”.

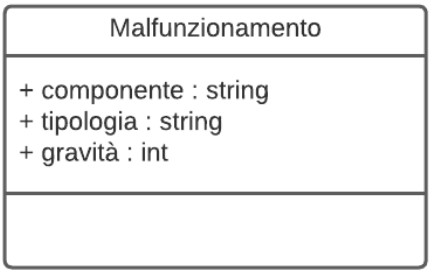


Per identificare le zone contenute nei piani di pulizia sono state definite le classi “Zona”, “Regione” e “Coordinata”. Una coordinata identifica un punto, una regione è un insieme di coordinate, mentre una zona contiene una regione, insieme a dati aggiuntivi che la riguardano, come il numero di rifiuti riconosciuti, i contenitori di rifiuti presenti nella zona, la lista di rifiuti riconosciuti dagli operatori ed i rifiuti non riconosciuti. I contenitori, le cui coordinate sono memorizzate nella classe, sono utilizzati dai robot per depositare i rifiuti nell’apposita categoria.



Il gestore ha un metodo “aggiorna_statistiche”, il quale si occupa di memorizzare le statistiche pubbliche relative ad una organizzazione tramite la classe “Gestore statistiche”. Questa operazione viene eseguita periodicamente ogni giorno. Per rendere ciò possibile è stata identificata una classe “Statistiche”, contenente informazioni riguardo una determinata regione e fascia oraria.

Segnalazione malfunzionamento



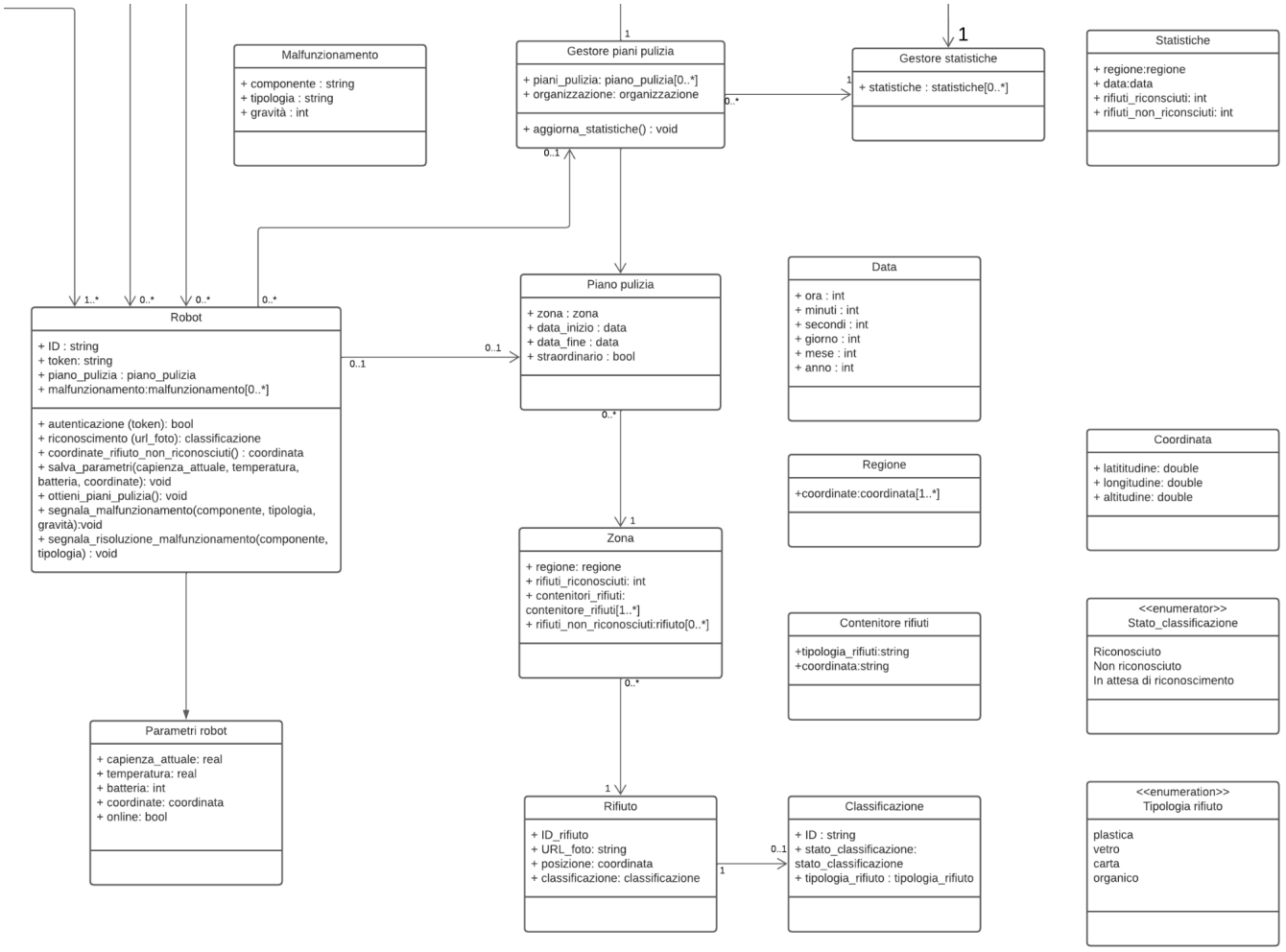
Allo scopo di ricevere segnalazioni di malfunzionamento da parte del robot, è stata creata la classe “Malfunzionamento”, la quale descrive il componente affetto, la tipologia di malfunzionamento ed un livello di gravità. La lista completa dei malfunzionamenti è contenuta all’interno della classe Robot.

Risoluzione malfunzionamento

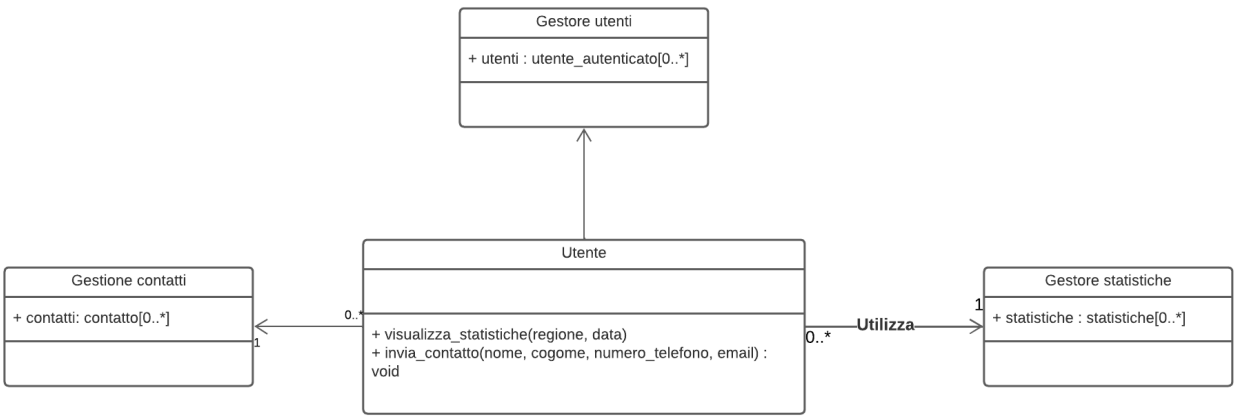
Una volta che un malfunzionamento viene risolto, il robot comunica l’operazione al sistema tramite il metodo “segnala_risoluzione_malfunzionamento”.

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi che implementano i requisiti funzionali relativi al robot.



1.2 Utente



La classe “Utente” raggruppa le funzionalità che sono comuni a tutti gli utenti.

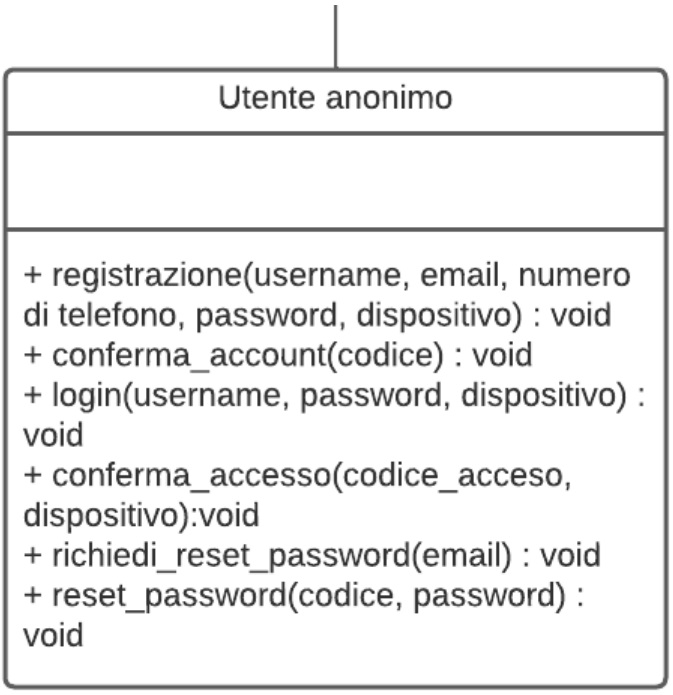
Visualizza statistiche

Al fine di visualizzare le statistiche, l'utente sfrutta la classe "Gestore statistiche".

Invia contatto

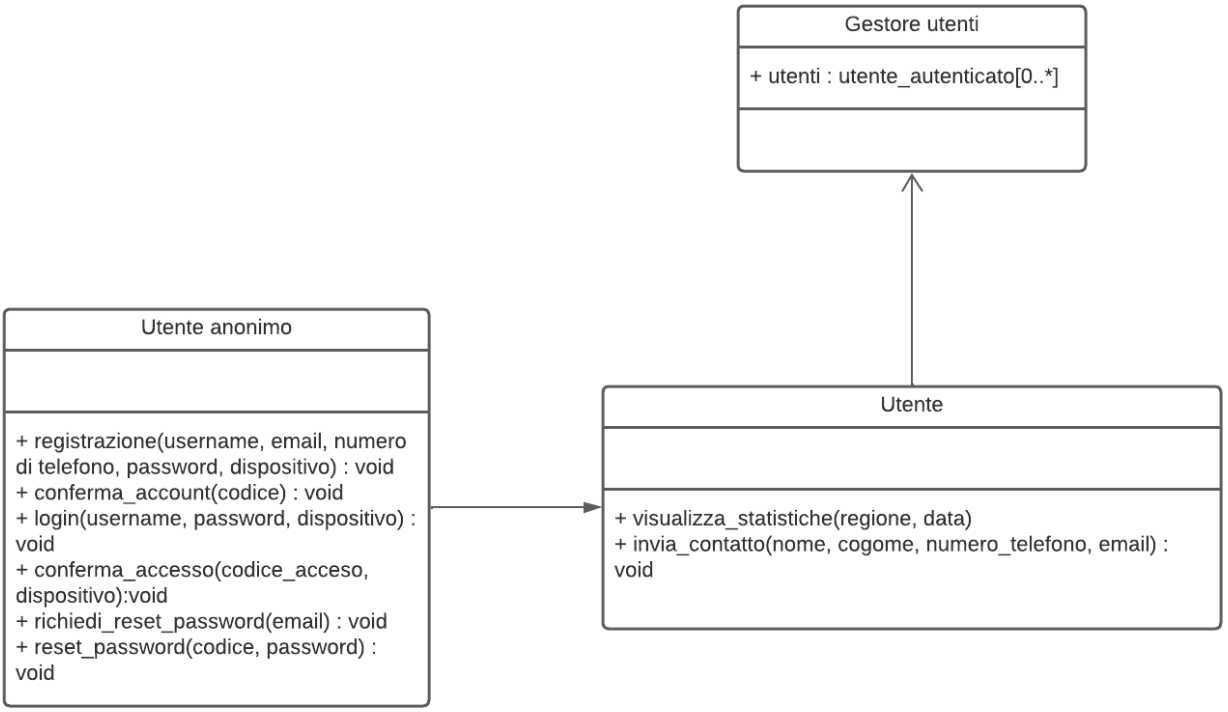
Per permettere all'utente di contattare il gestore del servizio, è stata definita una classe "Gestore contatti", fornendo le proprie informazioni (nome, cognome, numero di telefono ed email).

1.3 Utente anonimo

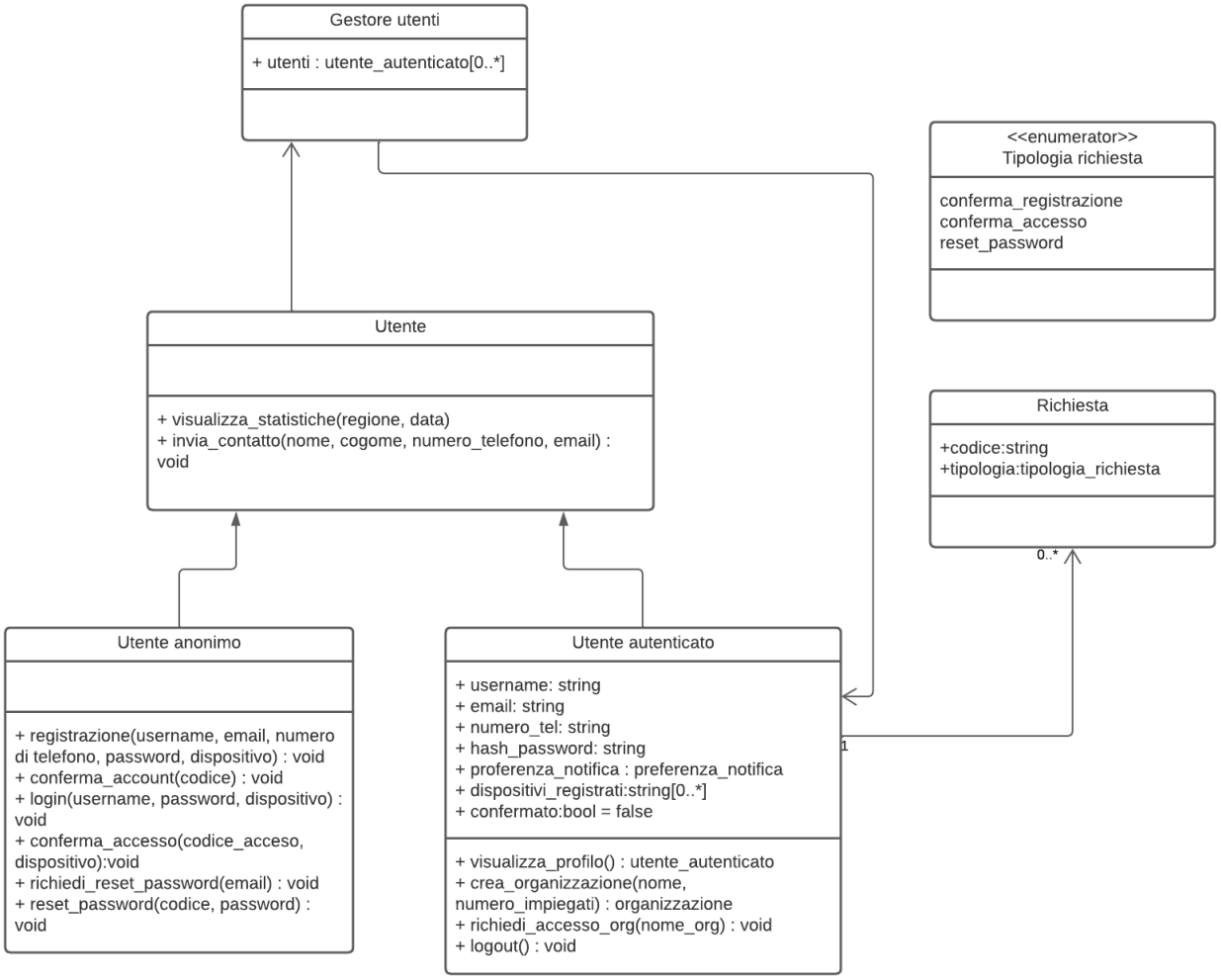


La classe "Utente anonimo" viene utilizzata dagli utenti non autenticati.

Registrazione

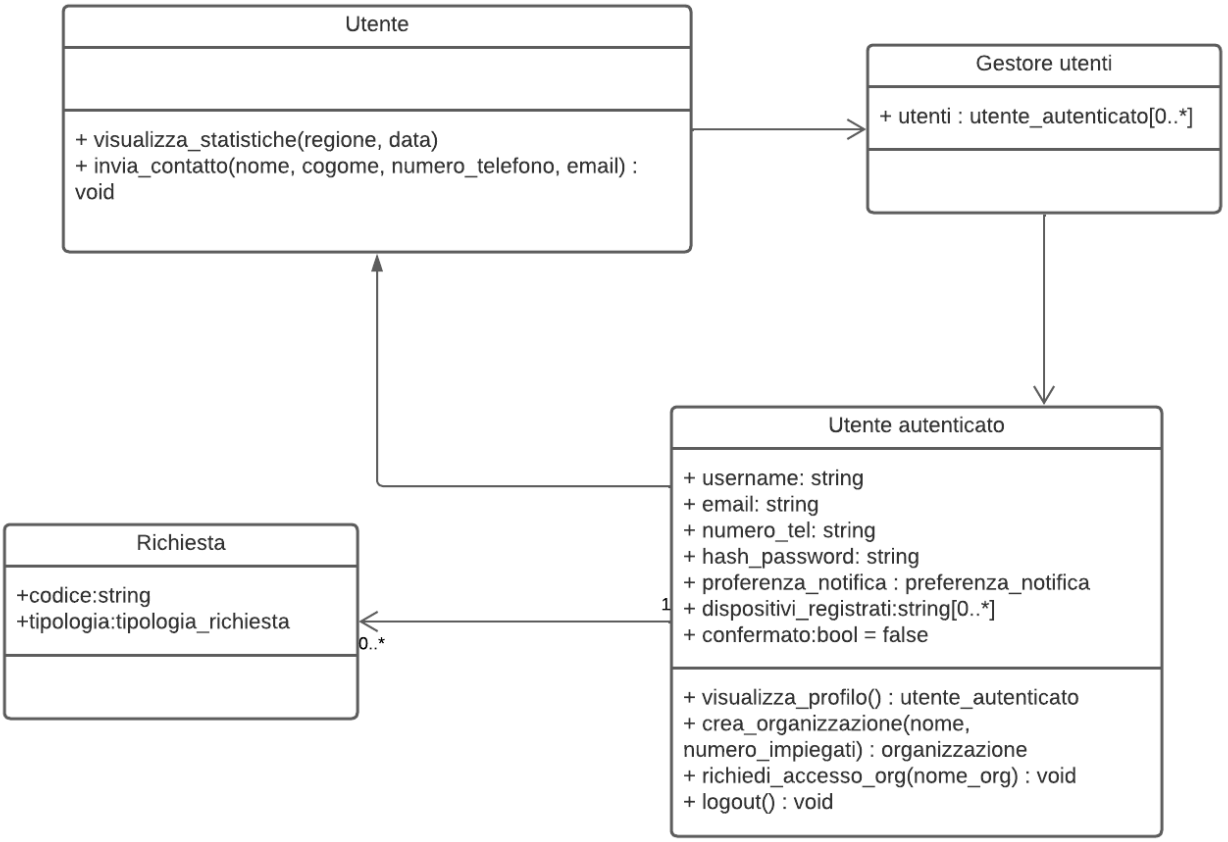


L’utente esegue la registrazione fornendo username, email, numero di telefono e password. In seguito riceverà una mail per conferma la validità del proprio account.



Una volta completata la registrazione, l'utente verrà inserito all'interno della lista di utenti contenuta in "Gestore utenti". In seguito, viene generata una richiesta con un codice associato all'account che si è registrato, allo scopo di confermare l'account. Allo scopo di implementare tale funzione è stata individuata una classe generica "Richiesta" di tipologia "conferma_registrazione".

Conferma account



Una volta registrato, l'utente dovrà aprire un link inviato per mail, il quale conterrà un codice univoco utilizzato per verificare l'account di quello specifico utente.

Login

L'utente effettua l'accesso al servizio fornendo email e password.

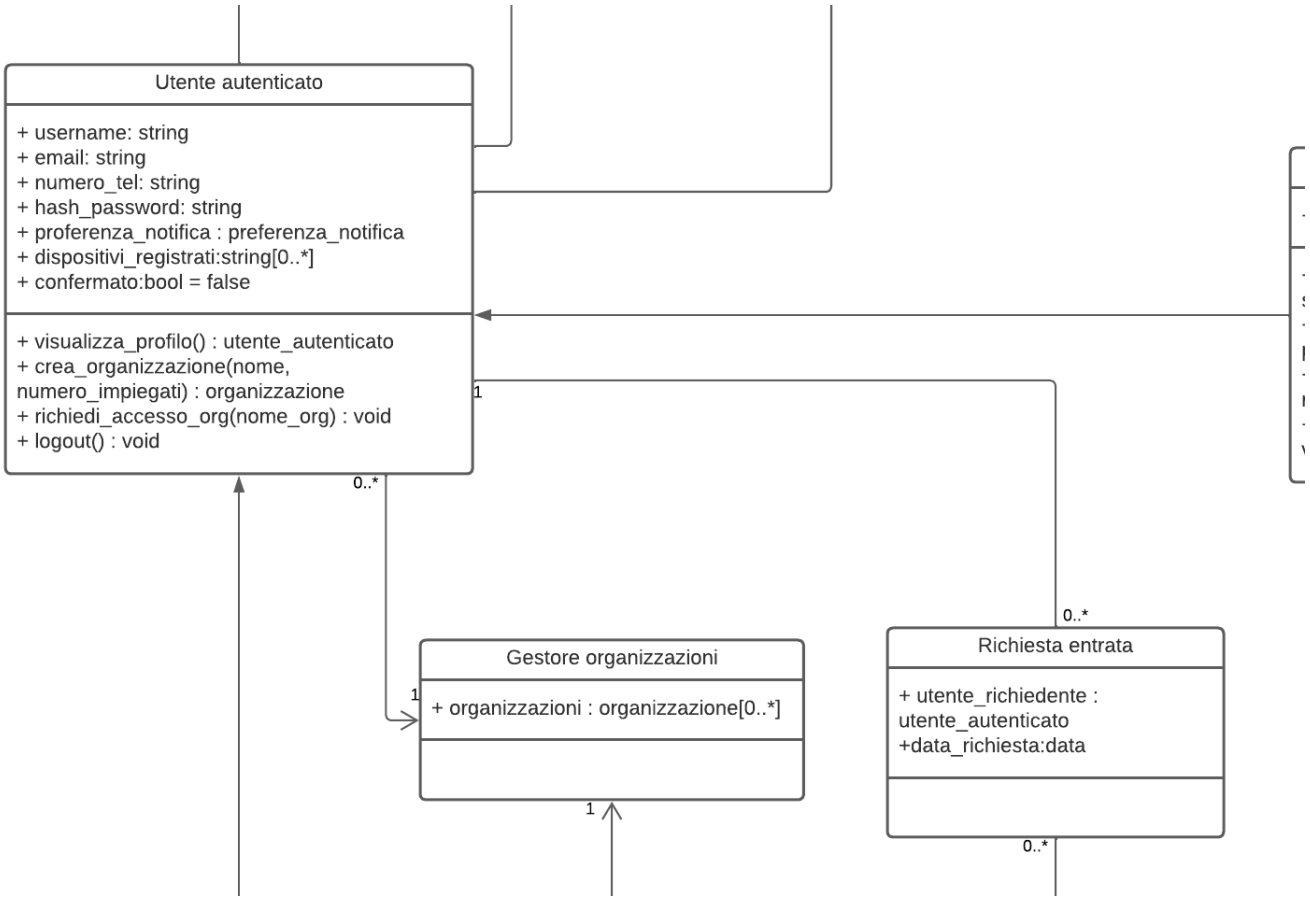
Conferma accesso

Nel caso in cui l'utente effettui l'accesso da un dispositivo nuovo, il sistema invierà una mail con un codice monouso per confermare l'identità dell'utente. L'utente dovrà inserire tale codice nella pagina di autenticazione a 2 passaggi.

Reset password

Nel caso l'avesse dimenticata, l'utente può richiedere il reset della password, fornendo la propria email. Ricevuta la mail, l'utente potrà aprire una nuova pagina tramite un link, dove inserirà la nuova password.

1.4 Utente autenticato



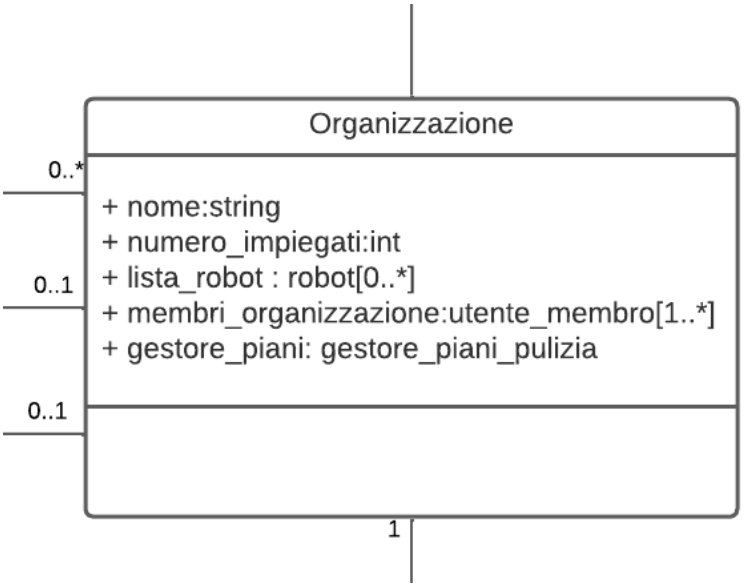
Come si evince dal nome, la classe "Utente autenticato" gestisce gli utenti che hanno effettuato l'accesso al servizio con un account valido.
La password dell'utente non viene memorizzata in chiaro, invece ne viene salvato l'hash SHA256. All'interno della classe vengono inoltre memorizzati la preferenza di notifica dell'utente (email o sms) e la lista di dispositivi che l'utente ha utilizzato per accedere con il proprio account.
Infine, ogni utente autenticato possiede una flag "confermato" per verificare se tale utente ha effettuato la verifica dell'account o no.

Visualizza profilo

L’utente autenticato può richiedere di visualizzare le proprie informazioni personali.

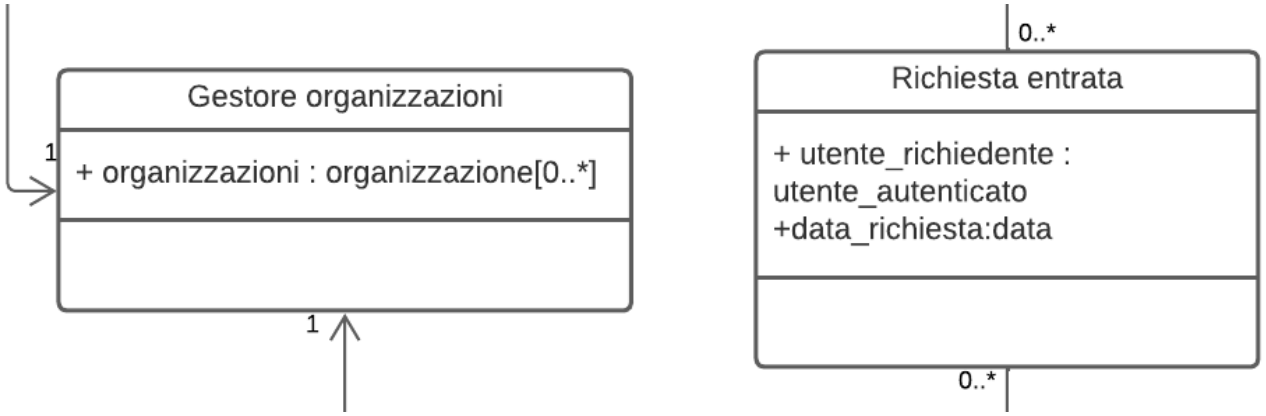
Crea organizzazione

Una volta autenticato, l'utente è in grado di creare una nuova organizzazione, perciò è stata identificata la classe “Organizzazione”. Una volta creata l’organizzazione, l’utente autenticato ne diventerà automaticamente amministratore primario.



Per creare un’organizzazione è necessario inserirne il nome ed il numero di impiegato (verrà richiesto un range, non il numero esatto). L’organizzazione contiene la lista di robot associati, oltre ai membri che ne fanno parte. Infine, ogni organizzazione possiede un gestore di piani di pulizia, il quale si occupa della memorizzazione ed associazione dei piani di pulizia ai robot posseduti dall’organizzazione.

Richiedi accesso ad un'organizzazione



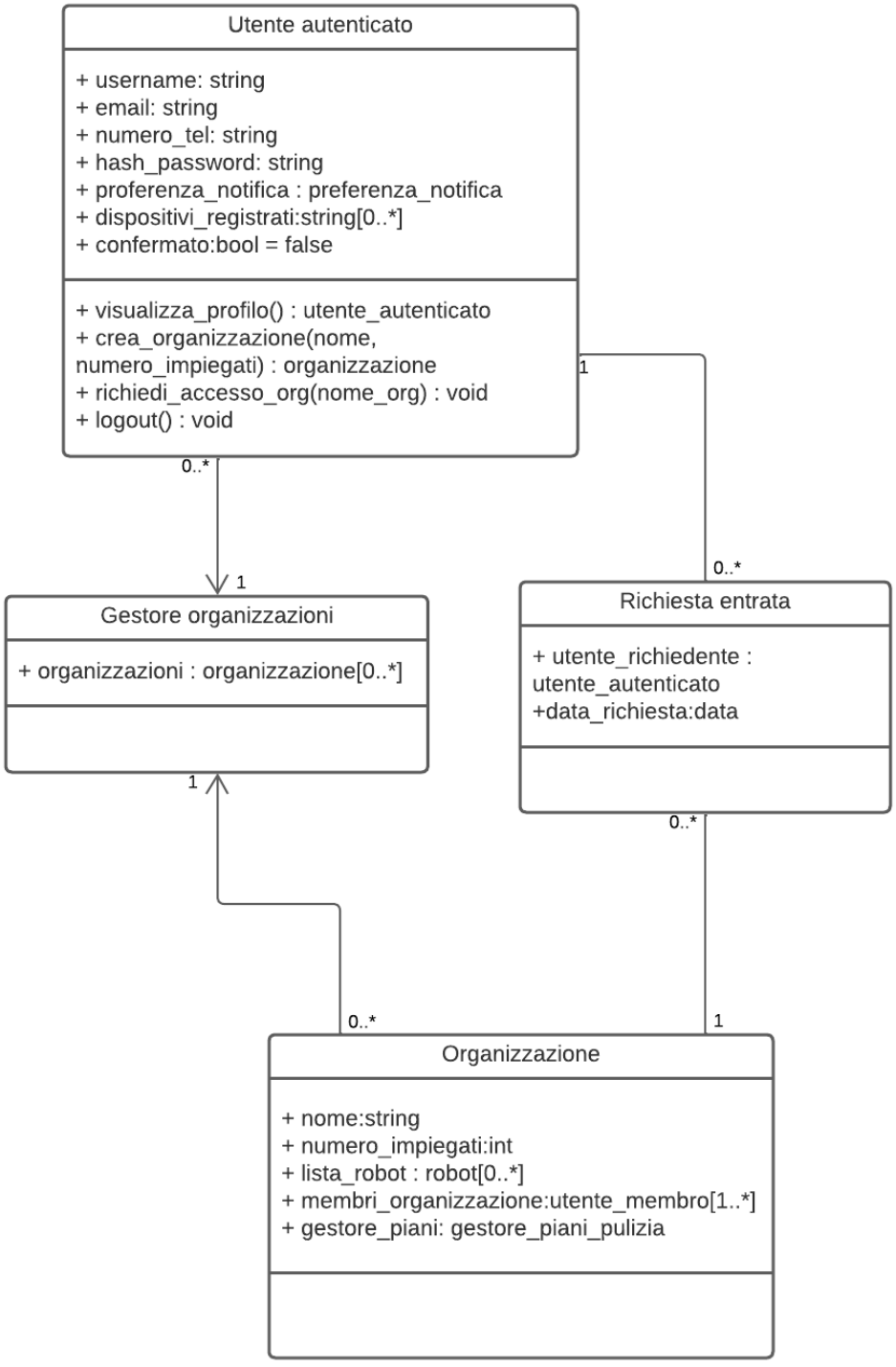
Oltre a creare un'organizzazione, l'utente autenticato può fare richiesta per entrare in un'organizzazione esistente. Per questo motivo sono state create le classi "Richiesta entrata" e "Gestore organizzazioni". La prima contiene le informazioni riguardo la richiesta, cioè l'organizzazione a cui l'utente vuole accedere ed il nome dell'utente richiedente. La classe Gestore organizzazione invece viene utilizzata per accedere a tutte le organizzazioni

Log out

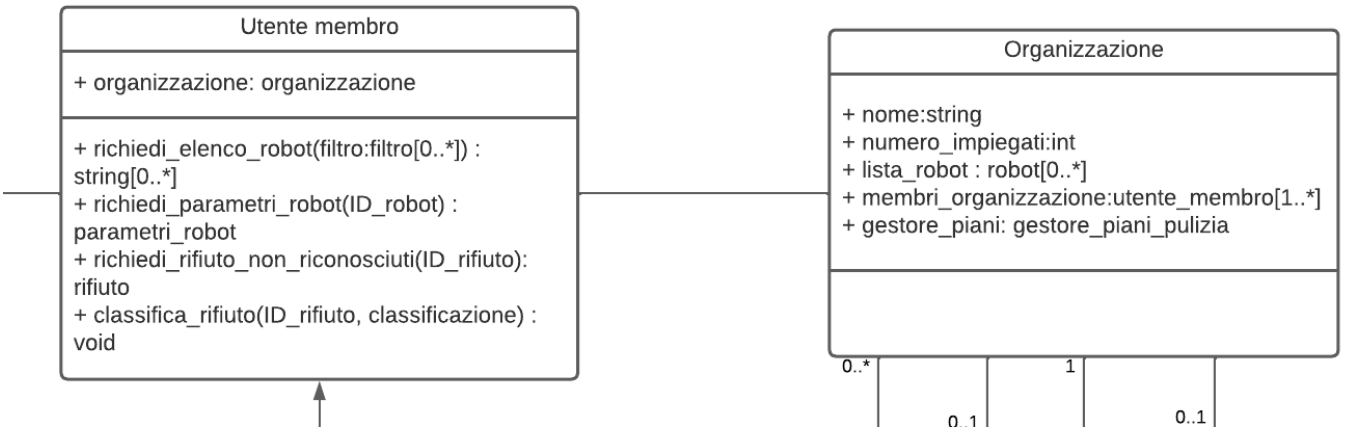
Allo scopo di terminare la sessione ed uscire dal proprio account, l'utente autenticato utilizza il metodo "logout".

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi che implementano i requisiti funzionali relativi all'utente autenticato.

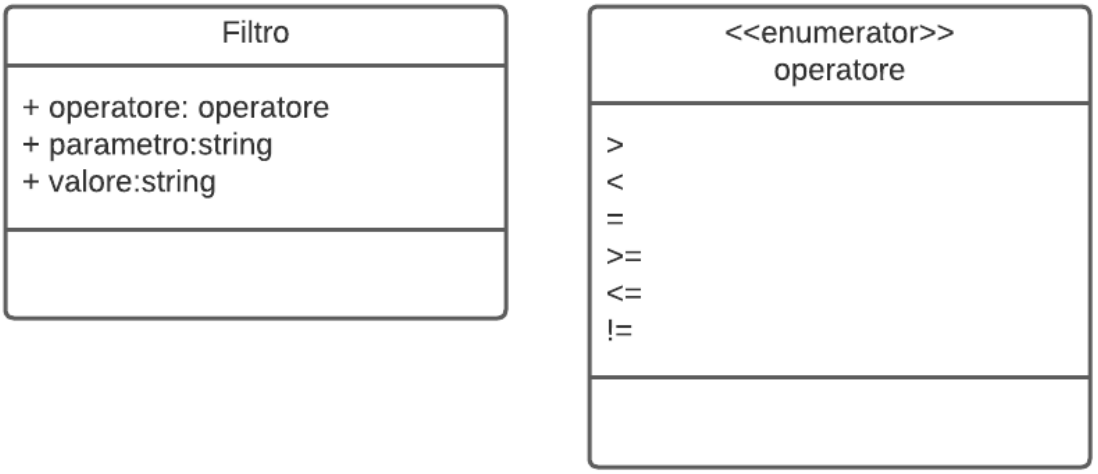

Utente autenticato
+ username: string
+ email: string
+ numero_tel: string
+ hash_password: string
+ proferenza_notifica : preferenza_notifica
+ dispositivi_registrati:string[0..*]
+ confermato:bool = false
+ visualizza_profilo() : utente_autenticato
+ crea_organizzazione(nome, numero_impiegati) : organizzazione
+ richiedi_accesso_org(nome_org) : void
+ logout() : void
1
0..*
1
0..*
Gestore organizzazioni
+ organizzazioni : organizzazione[0..*]
Richiesta entrata
+ utente_richiedente : utente_autenticato
+data_richiesta:data
1
0..*
0..*
1
Organizzazione
+ nome:string
+ numero_impiegati:int
+ lista_robot : robot[0..*]
+ membri_organizzazione:utente_membro[1..*]
+ gestore_piani: gestore_piani_pulizia

1.5 Utente Membro



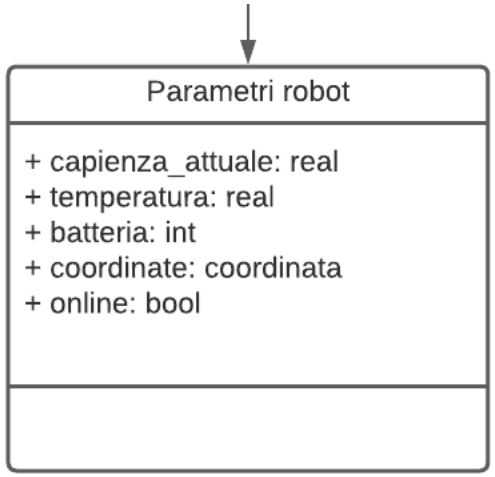
Gli utenti autenticati che entrano a far parte di un'organizzazione diventano "Utente membro", essendo ora legati ad una particolare organizzazione.

Elenco robot



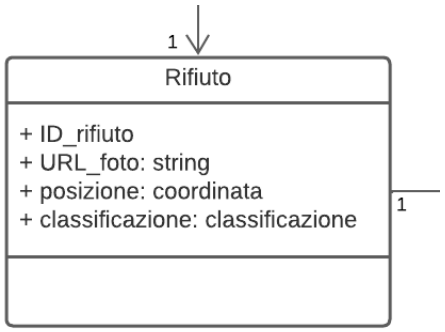
L'utente membro può richiedere l'elenco dei robot posseduti dalla propria organizzazione, applicando eventuali filtri sui parametri del robot (capienza attuale, temperatura, batteria, posizione, stato). Il filtro viene espresso nel formato <parametro robot> <operatore> <valore> ad esempio ("temperatura" > 20). Il metodo ritorna una lista di ID corrispondenti ai robot che soddisfano i filtri.

Parametri robot



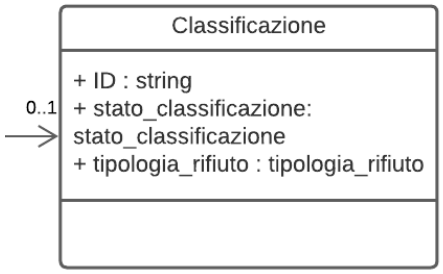
È possibile espandere i dettagli di un particolare robot, fornendone l'ID. Questo metodo fornisce i parametri del robot richiesto.

Richiedi rifiuti non riconosciuti



Quando un utente membro riceve una notifica di rifiuto non riconosciuto, è possibile accedere ai dettagli riguardanti la segnalazione tramite un link contenente l'ID del rifiuto.

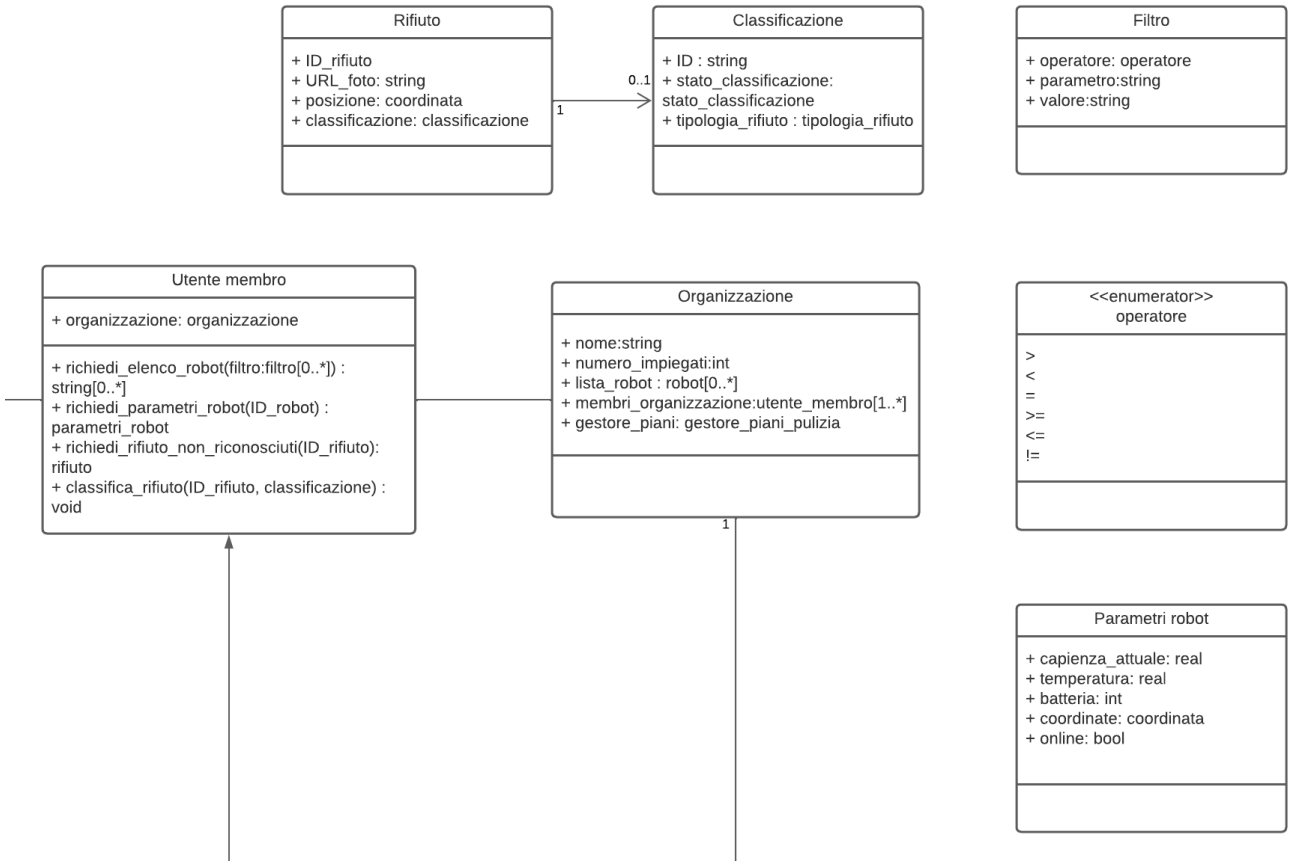
Classifica rifiuto



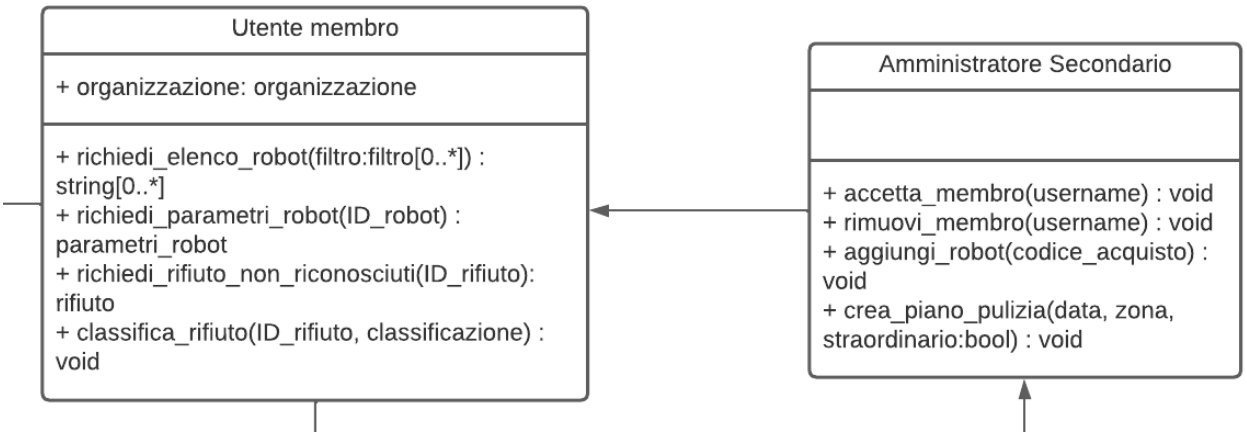
L'utente membro può classificare un rifiuto, scegliendo una delle categorie disponibili per la classificazione, oppure "Non riconoscibile" nel caso la foto del rifiuto non renda possibile l'identificazione. In questo caso, il prossimo robot che individua il rifiuto non riconosciuto e riscontra una classificazione "Non riconoscibile" invia una nuova foto al sistema.

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi che implementano i requisiti funzionali relativi all'utente membro.

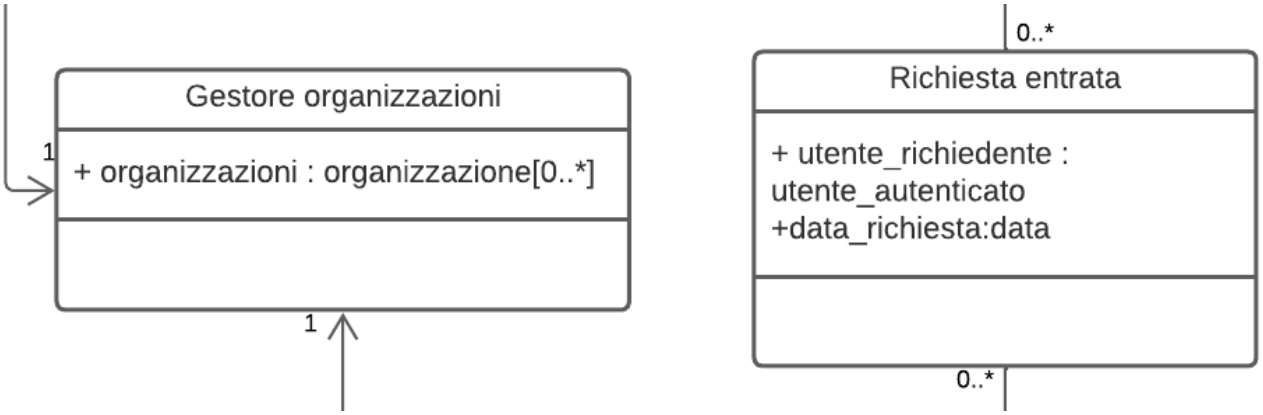


1.5 Amministratore secondario



Per gestire gli aspetti più importanti è stata ideata la classe di "Amministratore secondario".

Richieste di accesso

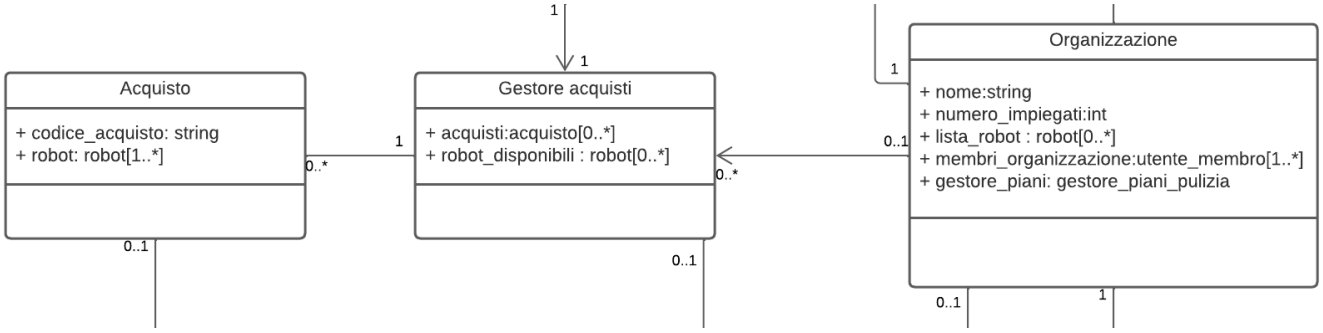


L'amministratore secondario può visualizzare le richieste di accesso all'organizzazione di cui fa parte. È quindi possibile accettare o rifiutare tali richieste.

Rimozione membri

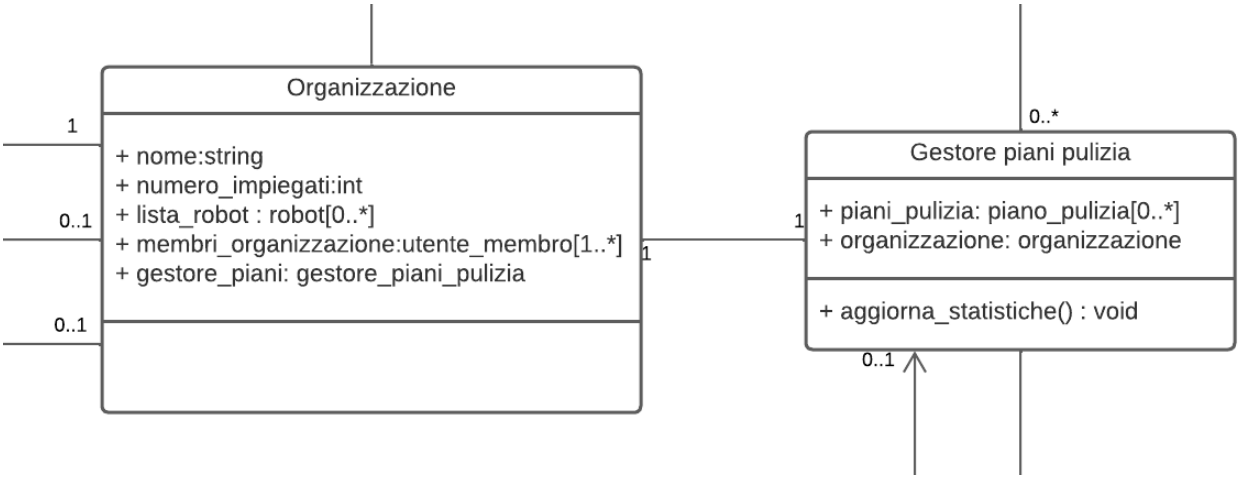
Gli utenti membro possono essere rimossi da un'organizzazione dall'amministratore secondario.

Aggiungere robot



Fornendo un codice d'acquisto valido, l'amministratore può aggiungere nuovi robot all'organizzazione. Per rendere ciò possibile, è stata identificata la classe "Gestore acquisti", contenente la lista di acquisti esistenti ed i robot disponibili, cioè i robot non legati ad alcuna organizzazione. Per rappresentare gli acquisti è stata creata la classe "Acquisto", identificata da un codice univoco e la lista di robot venduti.

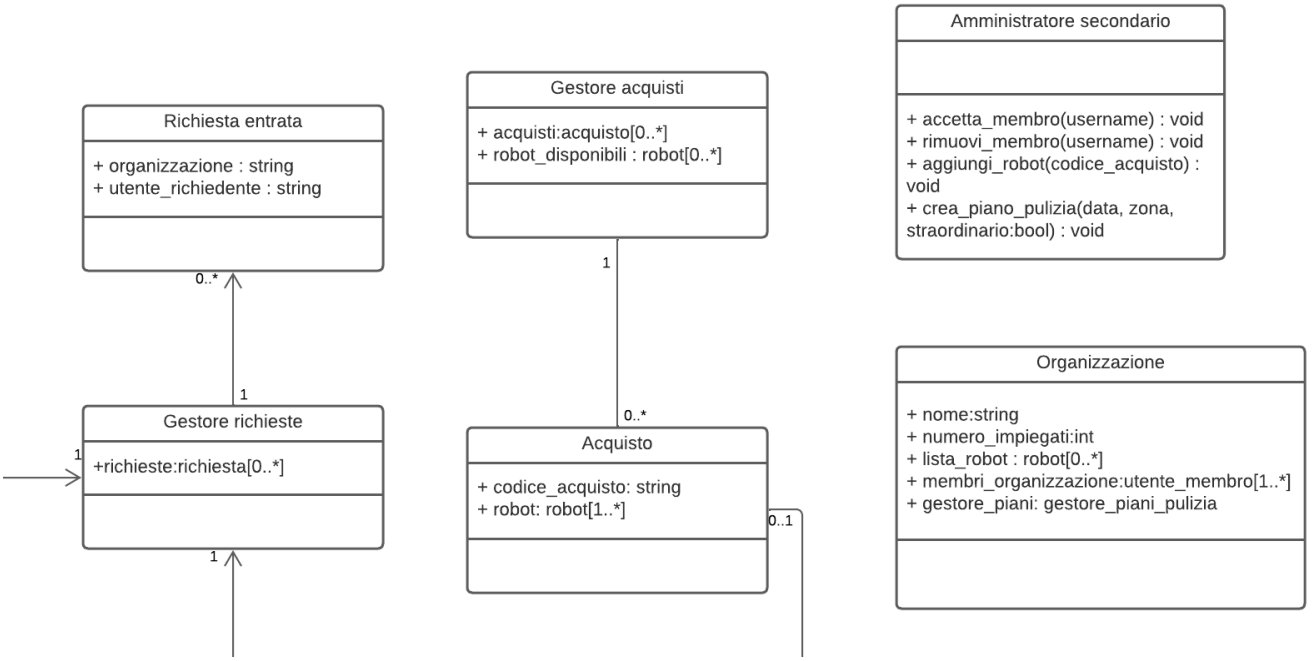
Creazione piani di pulizia



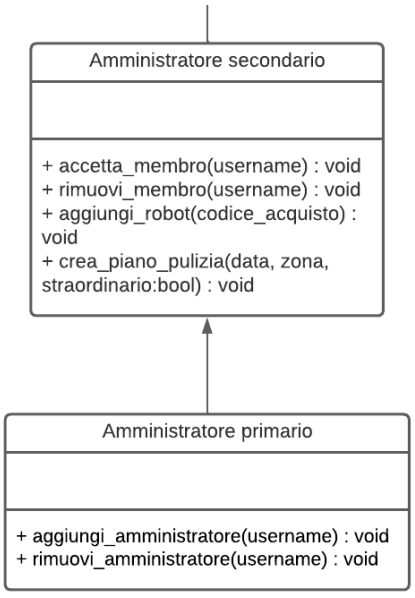
L'amministratore secondario può creare nuovi piani di pulizia utilizzando la classe Gestore piani pulizia. È necessario fornire una fascia oraria ed una zona, oltre alla possibilità di contrassegnare il piano di pulizia come straordinario, portandolo in cima alla coda di priorità.
Una volta creato, un piano di pulizia finisce all'interno della coda a priorità (contenuta dal Gestore piani pulizia), dove verrà prelevato dal primo robot disponibile, cioè un robot senza piano di pulizia.

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi che implementano i requisiti funzionali relativi all'amministratore secondario.



1.6 Amministratore primario



Quando un'organizzazione viene creata, l'utente che la crea ne diventa automaticamente "Amministratore primario". Questo tipo di utente possiede i più alti privilegi all'interno di un'organizzazione, ed ha la possibilità di fornire

ad altri utenti membri il privilegio di Amministratore secondario, oppure rimuoverli a chi già li possiede.

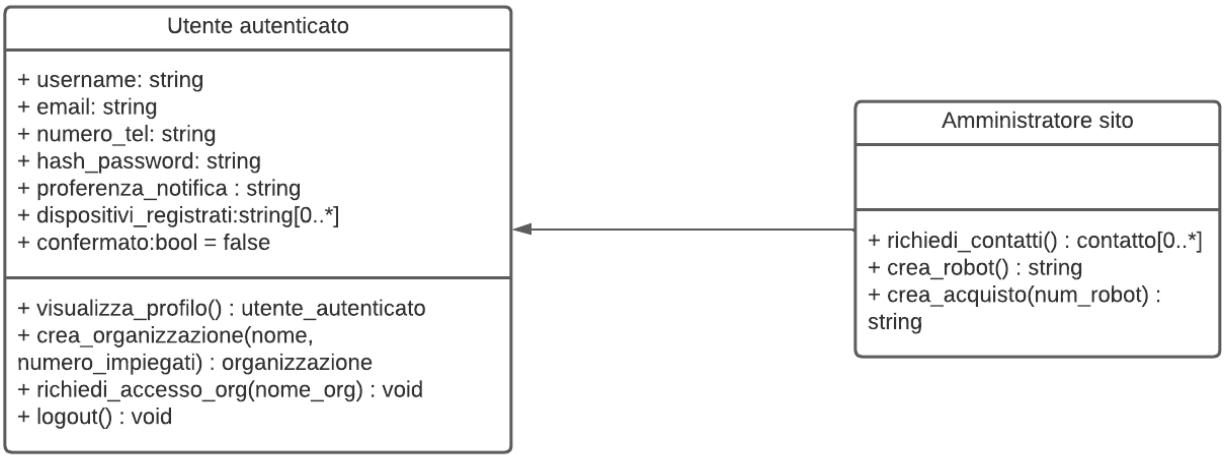
Aggiungere amministratore

L'amministratore primario può fornire permessi di amministratore secondario agli utenti membro.

Rimuovere amministratore

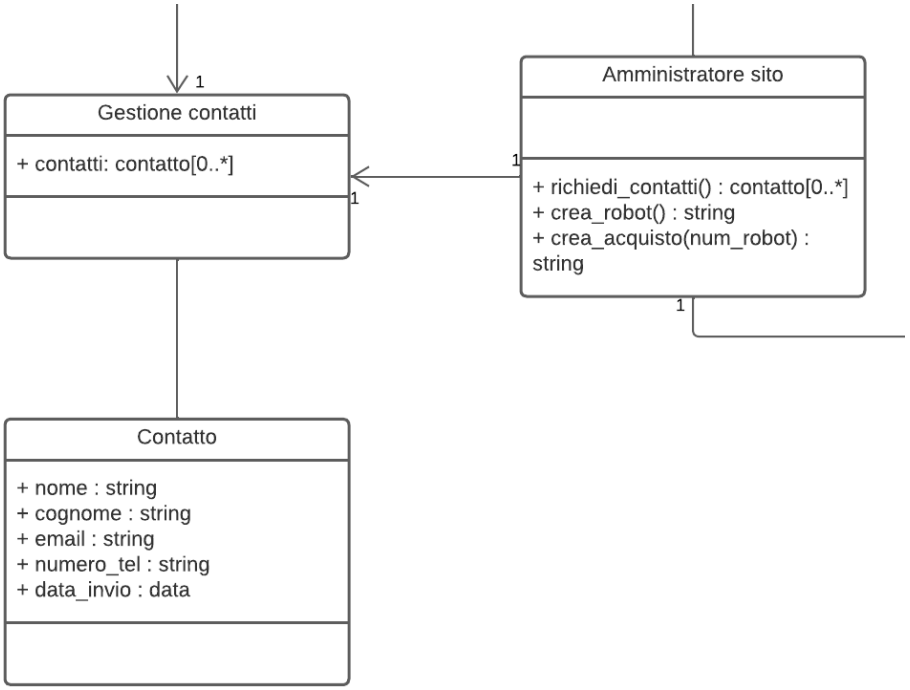
L'amministratore primario può togliere permessi di amministratore agli amministratori secondari, rendendoli nuovamente utenti membro.

1.7 Amministratore del sito



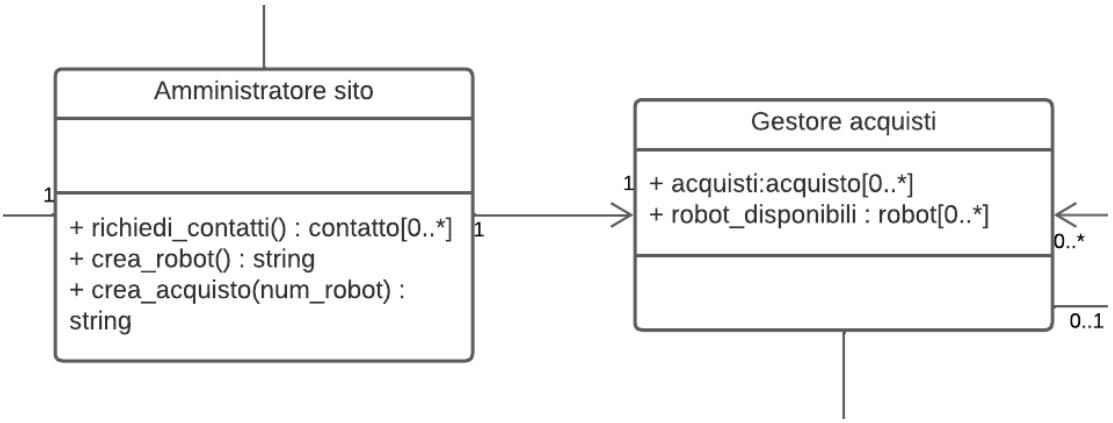
Infine, l'ultimo tipo di utente, "Amministratore del sito", responsabile della gestione e del corretto funzionamento del servizio.

Richiedere i contatti



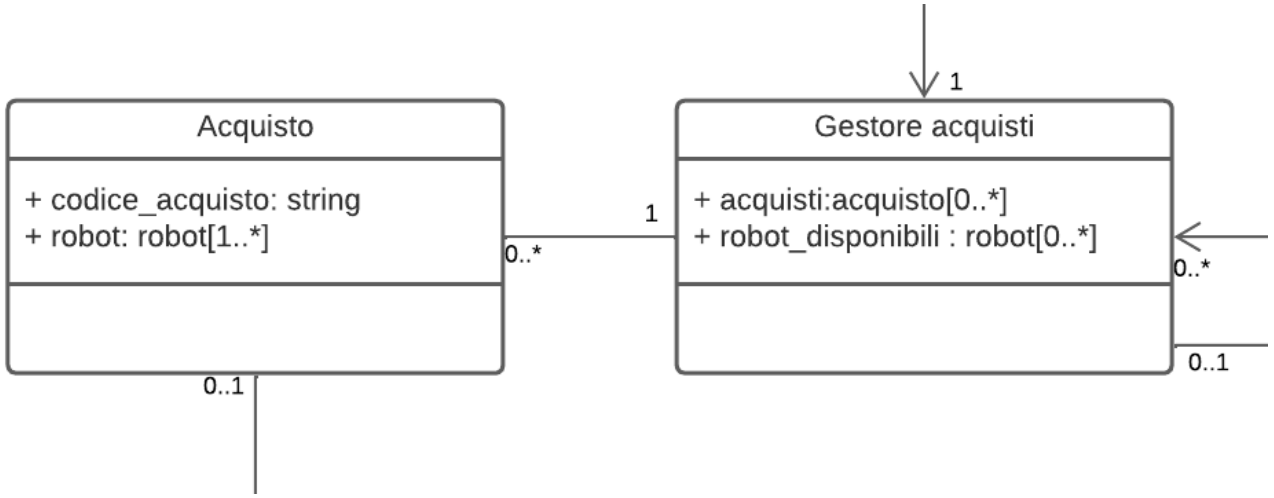
L'amministratore del sito può visualizzare le richieste di contatto da parte di potenziali clienti, utilizzando la classe Gestore contatti precedentemente definita. Una richiesta di contatto contiene informazioni relative al cliente (nome, cognome, email e numero di telefono) e la data di invio della richiesta.

Crea robot



Una delle funzionalità dell'amministratore del sito è quella di inserire nuovi robot all'interno del sistema, memorizzandoli all'interno del Gestore acquisti. Una volta creato un robot, verrà restituito il token di autenticazione all'amministratore, il quale provvederà ad inserirlo manualmente nel robot.
Infine, il robot viene memorizzato all'interno della lista di robot disponibili.

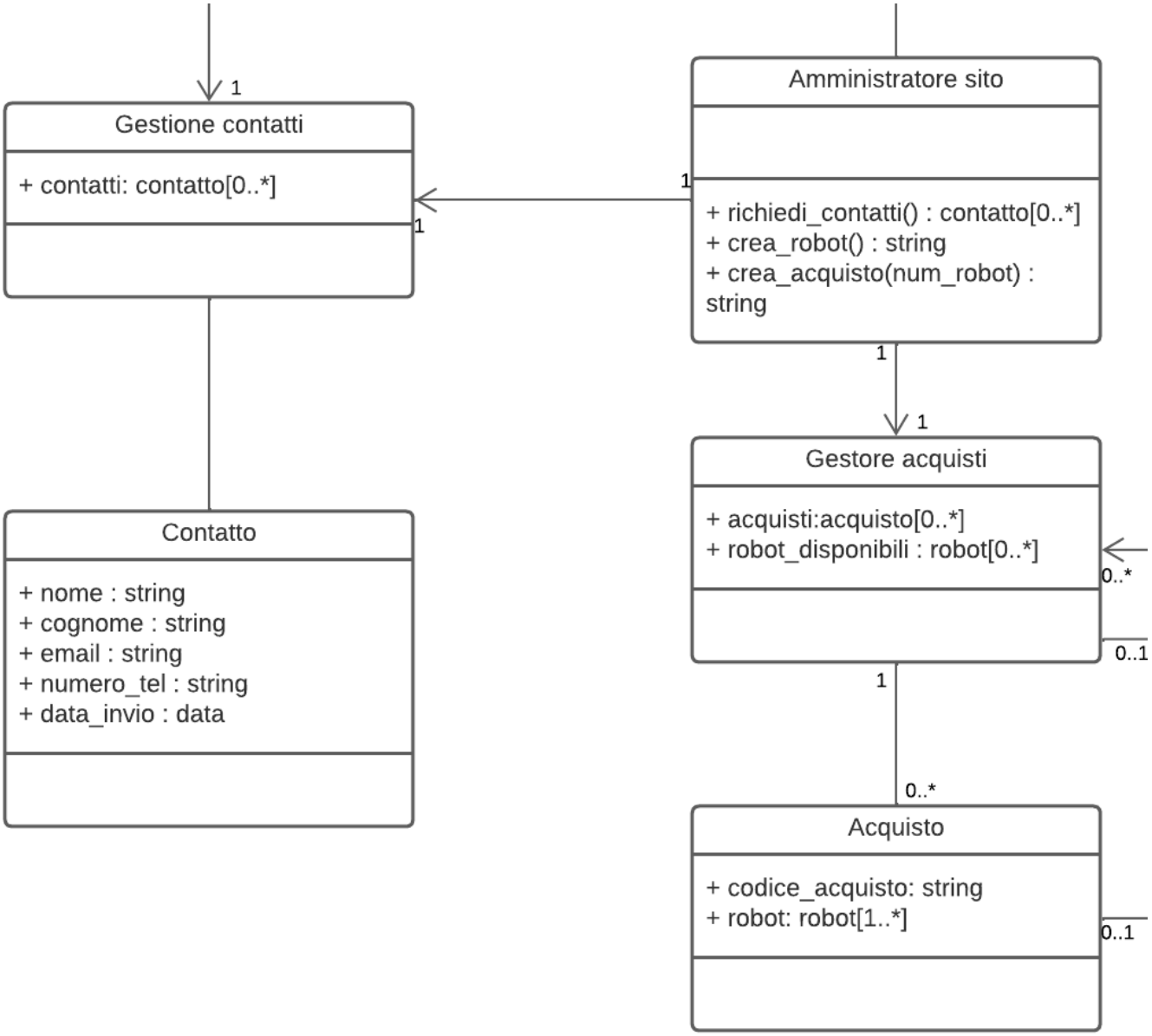
Crea acquisto



Per effettuare l'acquisto di uno o più robot, l'amministratore del sito crea degli acquisti che vengono memorizzati all'interno del Gestore acquisti. Tali acquisti contengono il codice relativo e i robot compresi nella transazione. Per completare l'operazione, l'organizzazione inserisce il codice d'acquisto ottenuto dall'amministratore per associare i robot.

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi che implementano i requisiti funzionali relativi all'amministratore del sito.

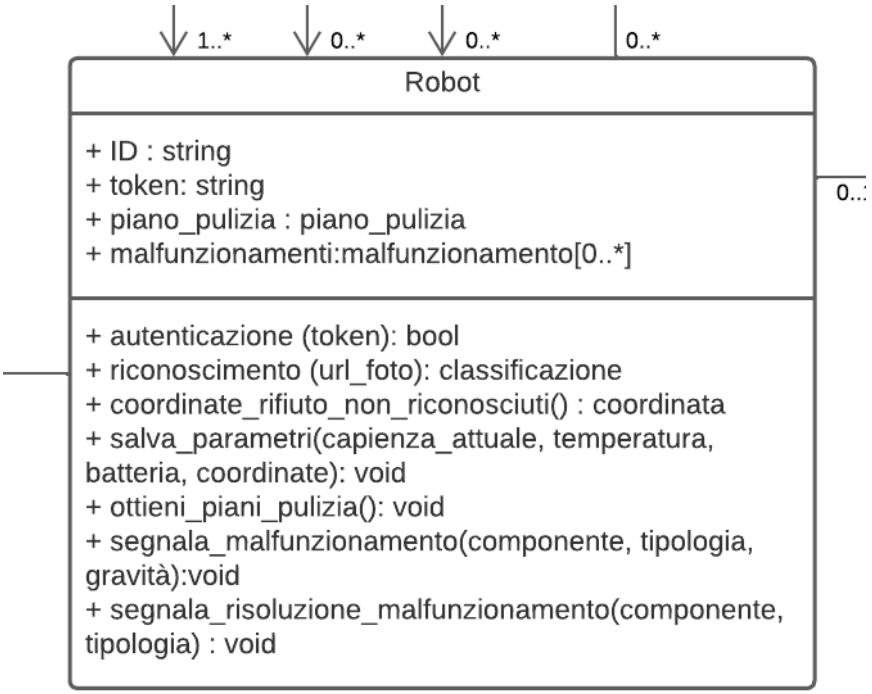


2. Codice in Object Constraint Language

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

2.1 Robot

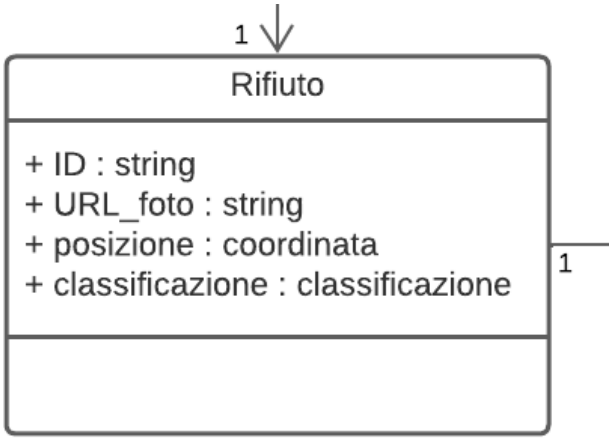
Autenticazione



Al fine di autenticarsi, il robot deve sempre possedere un ID e token validi. Espresso in OCL come:

```
context robot inv: self.token != "" AND self.ID != ""
```

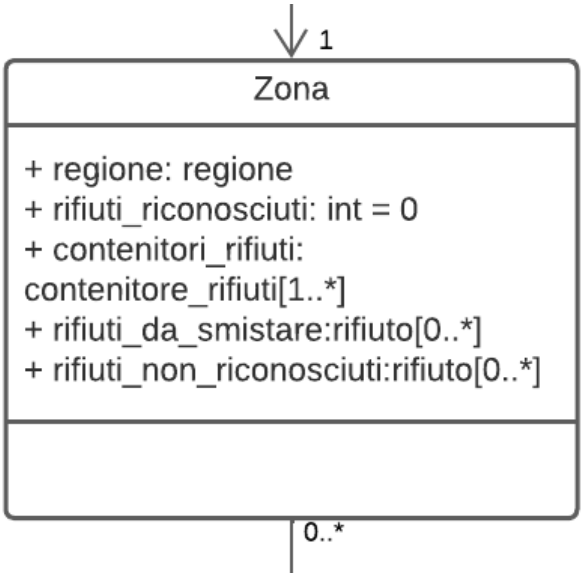
Validità rifiuto



Per essere considerato valido, un rifiuto deve avere un ID e un URL riferito alla foto del rifiuto.
Espresso in OCL come:

```
context rifiuto inv: self.ID != "" AND self.URL_foto != ""
```

Presenza contenitori nella zona



Ogni zona deve contenere almeno un contenitore per ogni tipologia di rifiuto.
Espresso in OCL come:

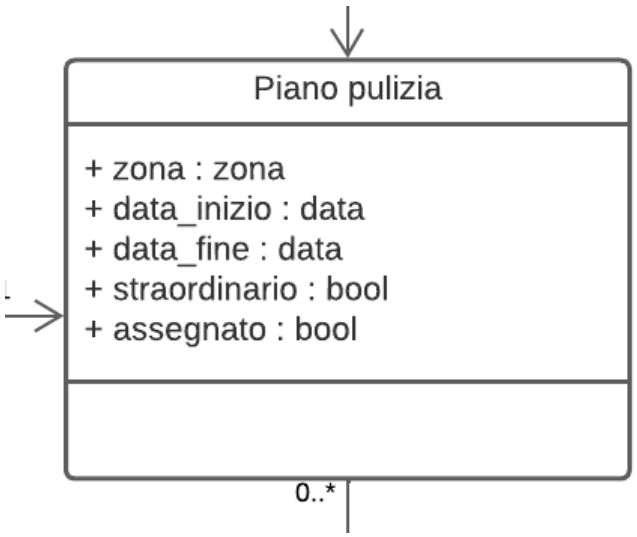
```
context zona
inv: self.contenitore_rifiuti->exists(c | c.tipologia = tipologia_rifiuto.plastica) AND
self.contenitore_rifiuti->exists(d | d.tipologia = tipologia_rifiuto.vetro) AND
self.contenitore_rifiuti->exists(e | e.tipologia = tipologia_rifiuto.carta) AND
self.contenitore_rifiuti->exists(f | f.tipologia = tipologia_rifiuto.organico)
```

Stato riconoscimento rifiuti non riconosciuti

Ogni rifiuto contenuto all'interno della lista di rifiuti non riconosciuti è in stato di "Attesa di riconoscimento".
Espresso in OCL come:

```
context zona
inv: !(self.rifiuti_non_riconosciuti->exists(c | c.classificazione.stato !=
stato_classificazione.in_attesa_di_riconoscimento))
```

Nuovo piano di pulizia



Prima di richiedere un nuovo piano di pulizia, il precedente deve essere stato completato, quindi la data di fine deve essere precedente alla data attuale.
Espresso in OCL come:

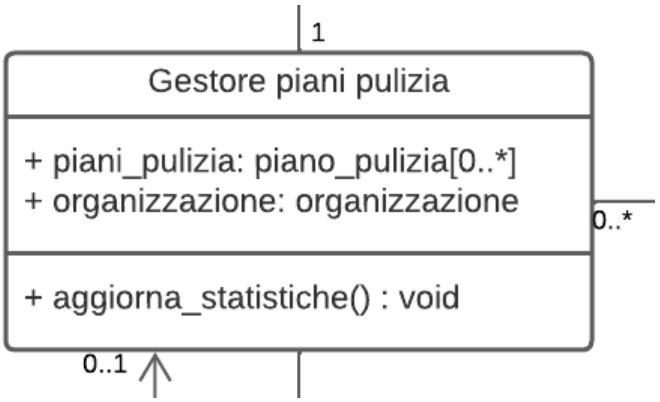
```
context robot::ottieni_piani_pulizia()
pre: self.piano_pulizia.data_fine <= data_attuale
```

Validità piano di pulizia

I piani ricevuti dal robot devono essere validi, in particolare il piano deve essere ancora in corso. Inoltre, il piano deve essere contrassegnato come "assegnato".
Espresso in OCL come:

```
context robot::ottieni_piani_pulizia()
post: self.piano_pulizia != null AND self.piano_pulizia.data_fine >= data_attuale
AND self.piano_pulizia.data_inizio <= data_attuale AND
self.piano_pulizia.assegnato = TRUE
```

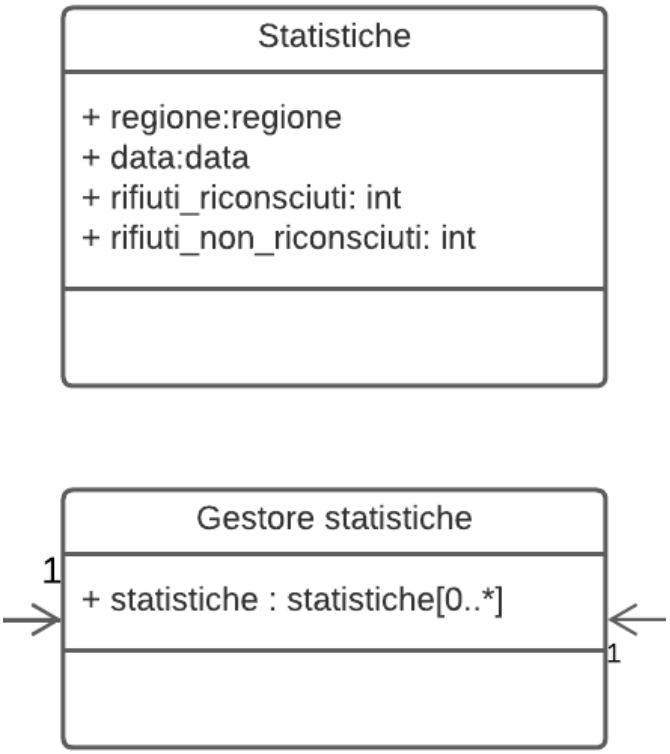
Assegnamento piani pulizia



I piani contenuti all'interno del gestore piani di pulizia non sono assegnati ad alcun robot.
Espresso in OCL:

```
context  gestione_piani_pulizia
inv: !(self.piani_pulizia->exists(c | c.assegnato = TRUE))
```

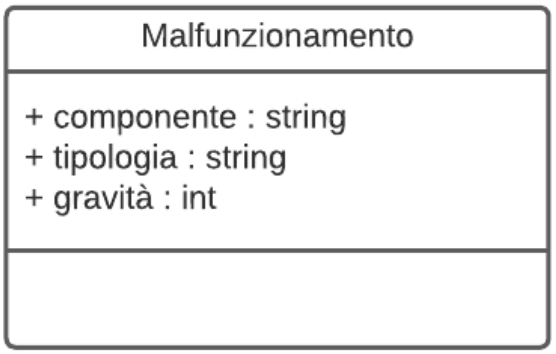
Raccoglimento statistiche periodico



Il gestore delle statistiche raccoglie le statistiche giornaliere di tutte le organizzazioni.
Espresso in OCL:

```
context gestore_statistiche:
inv: !self.statistiche->exists(c | data_attuale - c.data > 1)
```

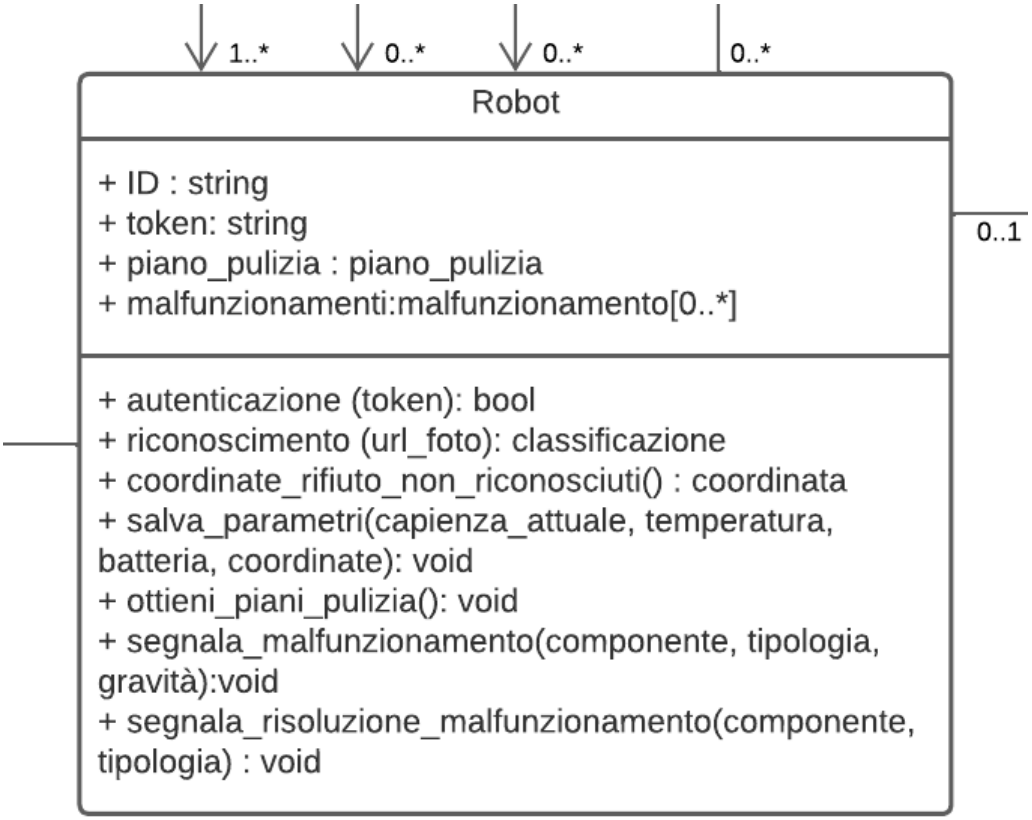
Livello di gravità del malfunzionamento



Il livello di gravità di un malfunzionamento può variare da 1 a 5.
Espresso in OCL:

```
context malfunzionamento inv: self.gravità >= 1 AND self.gravità <= 5
```

Risoluzione malfunzionamento



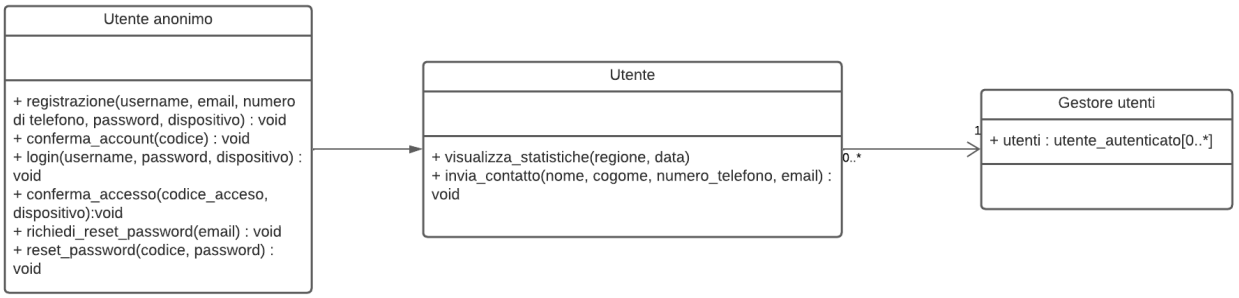
È necessario assicurarsi che il malfunzionamento che si vuole risolvere esista veramente. Inoltre, una volta risolto, il malfunzionamento deve essere rimosso.
Espresso in OCL come:

```
context robot::segnala_risoluzione_malfunzionamento(componente, tipologia)
pre: self.malfunzionamenti->exists(c | c.componente = componenente AND
c.tipologia = tipologia)
```

```
context robot::segnala_risoluzione_malfunzionamento(componente, tipologia)
post: !(self.malfunzionamenti->exists(c | c.componente = componenente AND
c.tipologia = tipologia))
```

2.2 Utente anonimo

Controllo unicità username



Per registrarsi, è necessario che l'utente inserisca un nome utente non ancora utilizzato.
Espresso in OCL come:

```
context  utente_anonimo::registrazione(username, email, numero_tel, password, dispositivo)
pre: !(self.gestore_utenti.utenti->exists(c | c.username = username))
```

Controllo unicità email

Per registrarsi, è necessario che l'utente inserisca un'email non ancora utilizzata.
Espresso in OCL come:

```
context  utente_anonimo::registrazione(username, email, numero_tel, password, dispositivo)
pre: !(self.gestore_utenti.utenti->exists(c | c.email = email))
```

Validità password

Per essere considerata valida, una password deve contenere almeno 8 caratteri e:

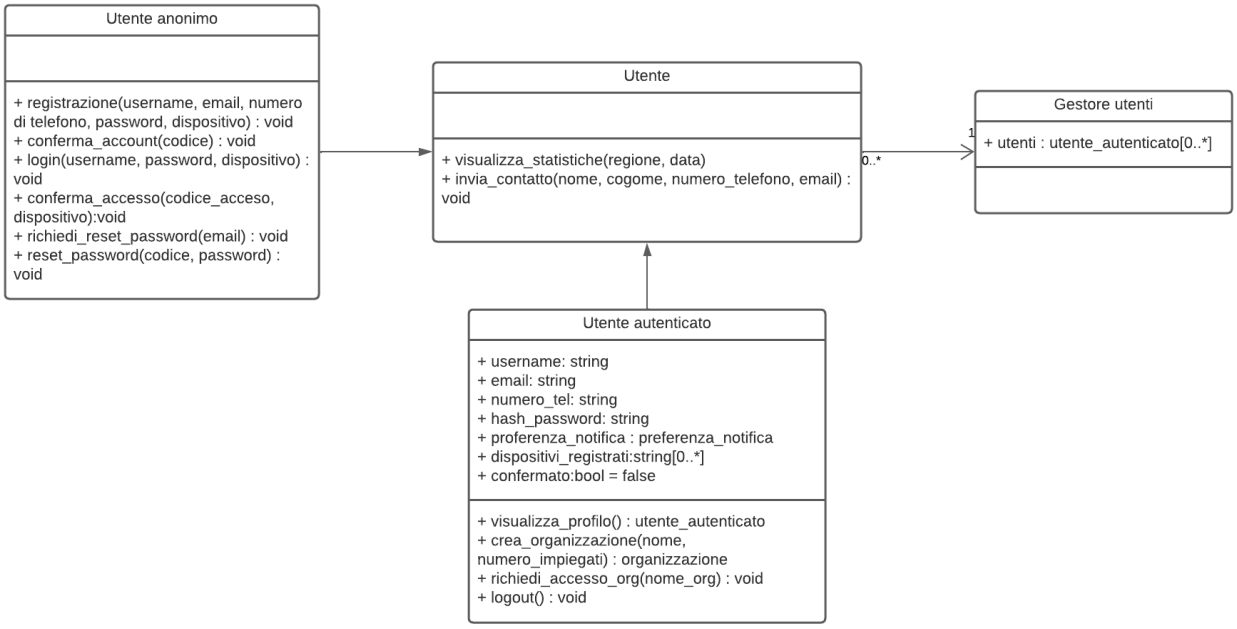
- contenere almeno un carattere maiuscolo (ASCII compresi da 'A' a 'Z')

- contenere almeno un carattere minuscolo (ASCII compresi da ‘a’ a ‘z’)
- contenere almeno un carattere speciale (ASCII compresi da ‘!’ a ‘/’ e da ‘:’ a ‘?’)

Espresso in OCL come:

```
context utente_anonimo::registrazione(username, email, numero_tel, password, dispositivo)
pre: password.size() >= 8 AND password.exists(c | c >= 'a' AND c <= 'z') AND password.exists(c | c >= 'A' AND c <= 'Z') AND password.exists(c | (c >= '!' AND c <= '/') OR (c >= ':' AND c <= '?'))
```

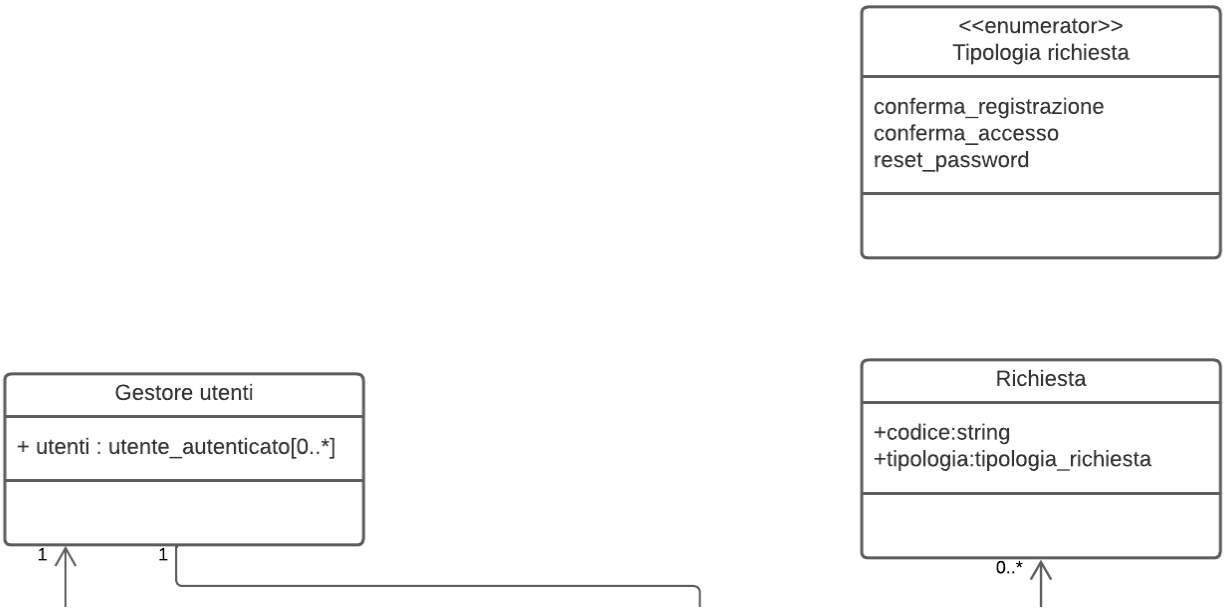
Aggiungi nuovo dispositivo registrato



Dopo aver completato la registrazione, il dispositivo dell’utente verrà memorizzato all’interno di una lista.
Espresso in OCL come:

```
context utente_anonimo::registrazione(username, email, numero_tel, password, dispositivo)
post: self.gestione_utenti.utenti->select(c | c.username =username).dispositivi_registrati->includes(dispositivo)
```

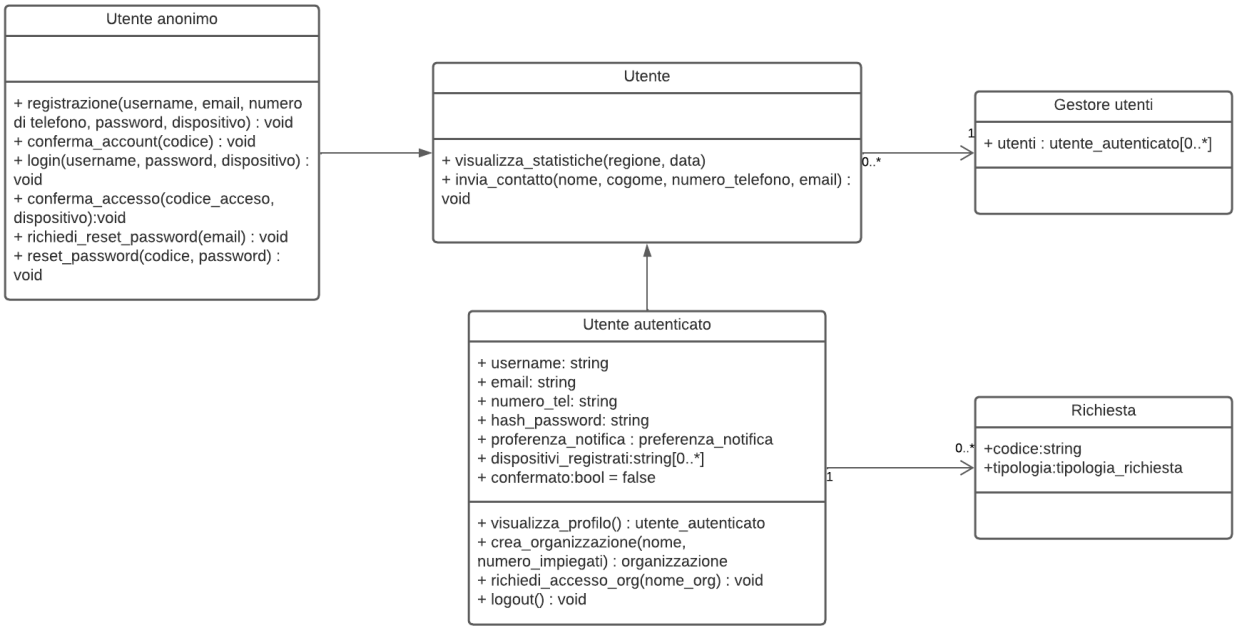
Cancellazione richieste di conferma



Dopo che l'utente effettua la conferma del proprio account, la richiesta relativa a tale conferma deve essere rimossa.
Espresso in OCL come:

```
context  utente_anonimo::conferma_account(codice)
post: !self.gestore_utenti.richieste->exists(c | c.codice = codice AND c.tipologia =
tipologia_richiesta.conferma_registrazione)
```

Controllo dispositivo durante il login



Durante la fase di login, il sistema controlla se il dispositivo utilizzato rientra nella lista di dispositivi registrati, ovvero la lista di dispositivi utilizzati dall'utente per usare il servizio. Nel caso in cui non venga riconosciuto, è necessario che venga creata una nuova richiesta per confermare l'accesso. Una volta confermato l'accesso, il dispositivo verrà aggiunto alla lista.
Espresso in OCL come:

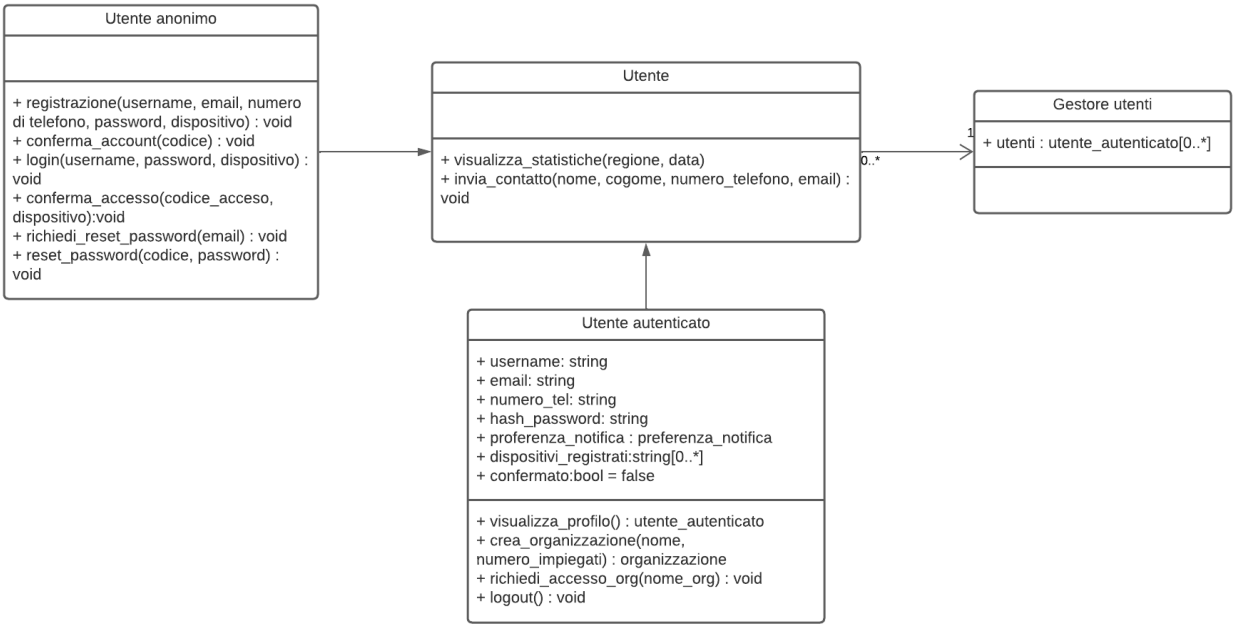
```
context  utente_anonimo::login(username, password, dispositivo)
post: if (!self.gestore_utenti.utenti->select(c | c.username =
username).dispositivi_registrati->includes(dispositivo)) then
(self.gestore_utenti.richieste->exists(d | d-utente.username = username AND
d.tipologia = tipologia_richiesta.conferma_registrazione)) end-if
```

Cancellazione richiesta di accesso

Dopo aver confermato l'accesso, la richiesta di tipo "conferma_accesso" relativa all'utente va rimossa.
Espresso in OCL come:

```
context  utente_anonimo::conferma_accesso(codice)
post: !self.gestore_utenti.richieste->exists(c | c.codice = codice AND c.tipologia =
tipologia_richiesta.conferma_accesso)
```

Reset nuova password



Per effettuare con successo il reset della password, l'utente deve inserire una password diversa da quella precedente.
Espresso in OCL come:

```
context  utente_anonimo::reset_password(codice, password)
pre: hash(password) != self.gestore_utenti.richieste->select(c | c.codice =
codice).utente.hash_password
```

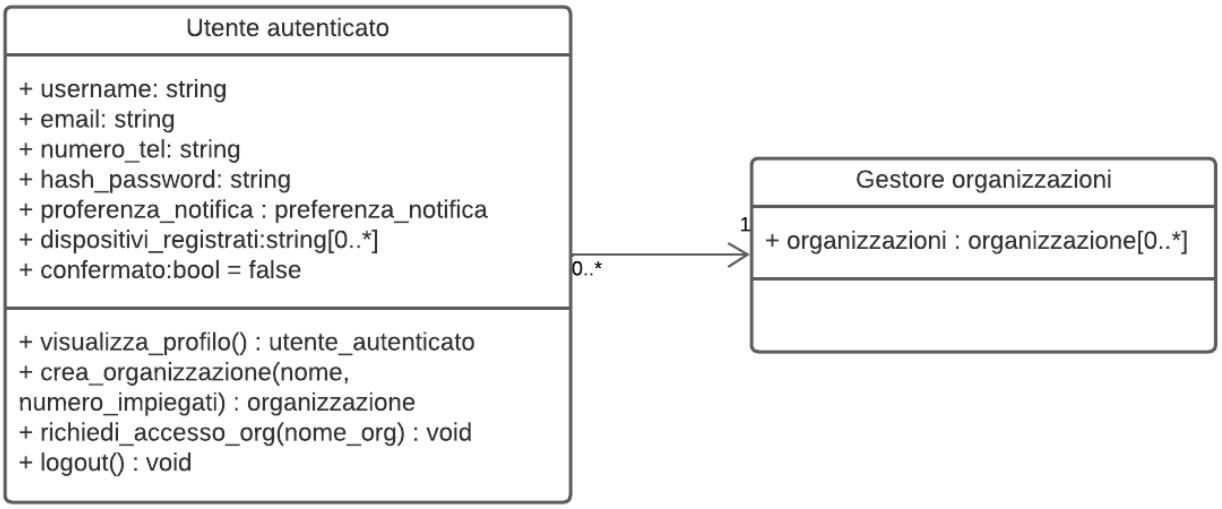
Cancellazione richiesta reset password

Dopo aver effettuato il reset della password, la richiesta di tipo "reset_password" relativa all'utente va rimossa.
Espresso in OCL come:

```
context  utente_anonimo::reset_password(codice, codice)
post: !self.gestore_utenti.richieste->exists(c | c.codice = codice AND c.tipologia =
tipologia_richiesta.reset_password)
```

2.3 Utente autenticato

Unicità nome organizzazione



Per creare una nuova organizzazione, l'utente deve fornire un nome non ancora utilizzato.
Espresso in OCL come:

```
context  utente_autenticato::crea_organizzazione(nome, numero_impiegati)
pre:  !(self.gestore_organizzazioni.organizzazioni->exists(c| c.nome = nome))
```

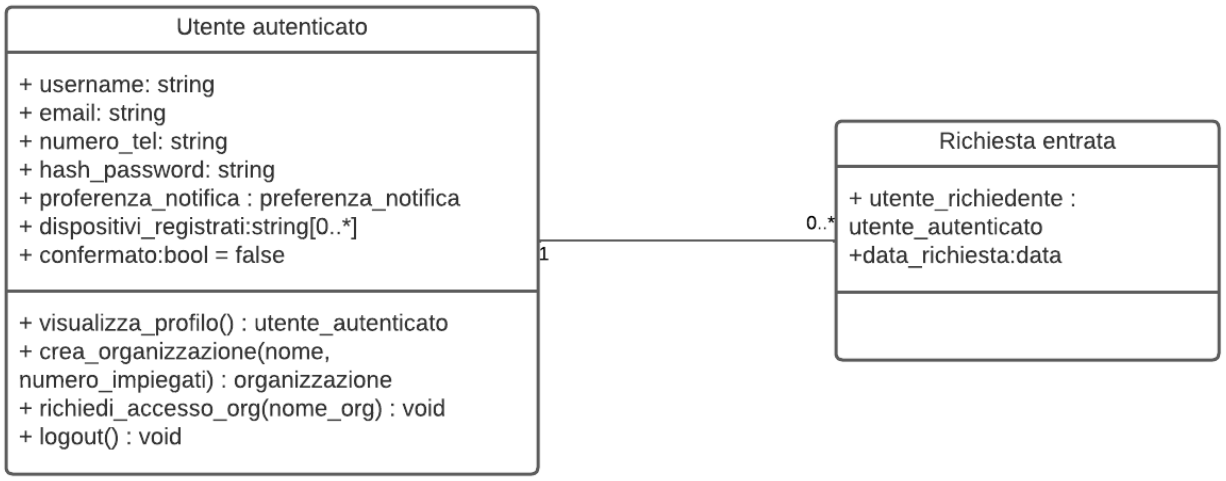
Richiedi accesso ad organizzazione esistente

L'utente deve fornire il nome di un'organizzazione esistente per fare richiesta di entrata.

Espresso in OCL come:

```
context  utente_autenticato::richiedi_accesso_org(nome_org)
pre: (self.gestore_organizzazioni.organizzazioni->exists(c| c.nome = nome_org))
```

Creazione richiesta di accesso

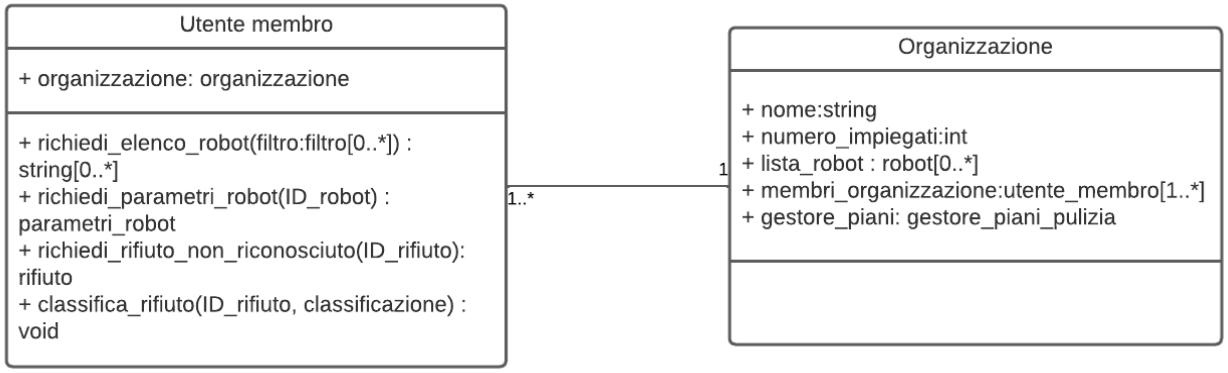


Dopo aver effettuato la richiesta, quest'ultima verrà memorizzata all'interno della lista "richieste_entrata".
Espresso in OCL come:

```
context  utente_autenticato::richiedi_accesso_org(nome_org)
post: self.richieste_entrata->exists(c | c.organizzazione.nome = nome)
```

2.4 Utente membro

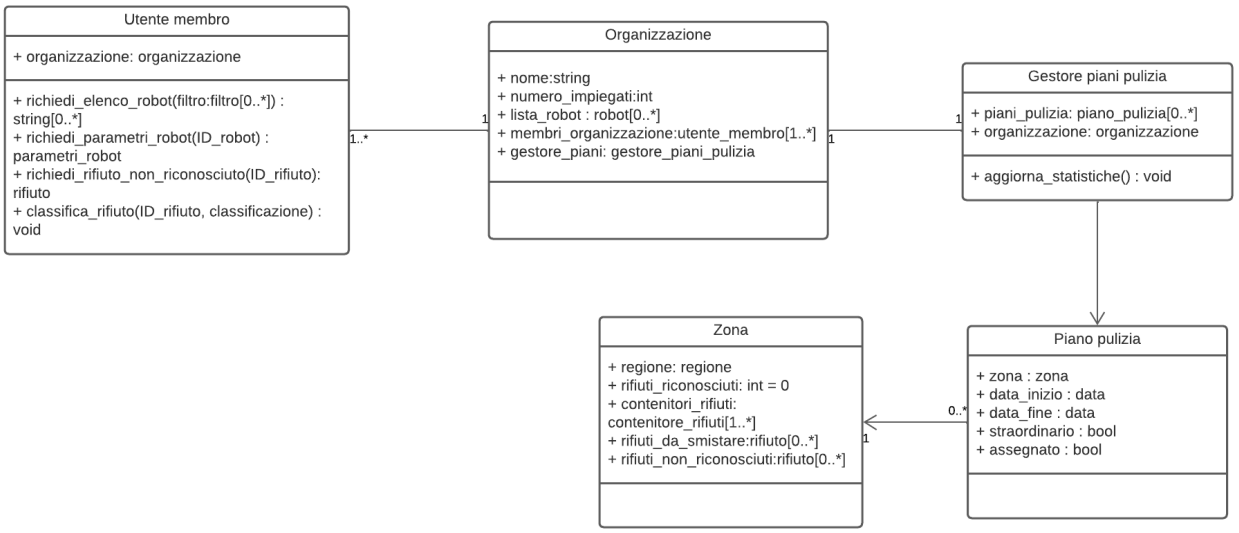
Esistenza robot



Per richiedere i parametri, è necessario che il robot esista.
Espresso in OCL come:

```
context  utente_membro::richiedi_parametri_robot(ID_robot)
pre:  self.organizzazione.lista_robot->exists(c | c.ID = ID_robot)
```

Esistenza rifiuto



Per richiedere i dettagli, è necessario che il rifiuto esista.

Espresso in OCL come:

```
context  utente_membro::richiedi_rifiuti_non_riconosciuto(ID_rifiuto)
pre:  self.organizzazione.gestore_piani_pulizia.piani_pulizia->exists(c |
c.zona.rifiuti_non_riconosciuti->exists(d | d.ID = ID_rifiuto))
```

Condizione di classificazione

Per classificare un rifiuto è necessario che l'ID rifiuto corrisponda ad un rifiuto esistente.
Espresso in OCL come:

```
context  utente_membro::classifica_rifiuto(ID_rifiuto, classificazione)
pre:  self.organizzazione.gestore_piani_pulizia.piani_pulizia->exists(c |
c.zona.rifiuti_non_riconosciuti->exists(d | d.ID = ID_rifiuto))
```

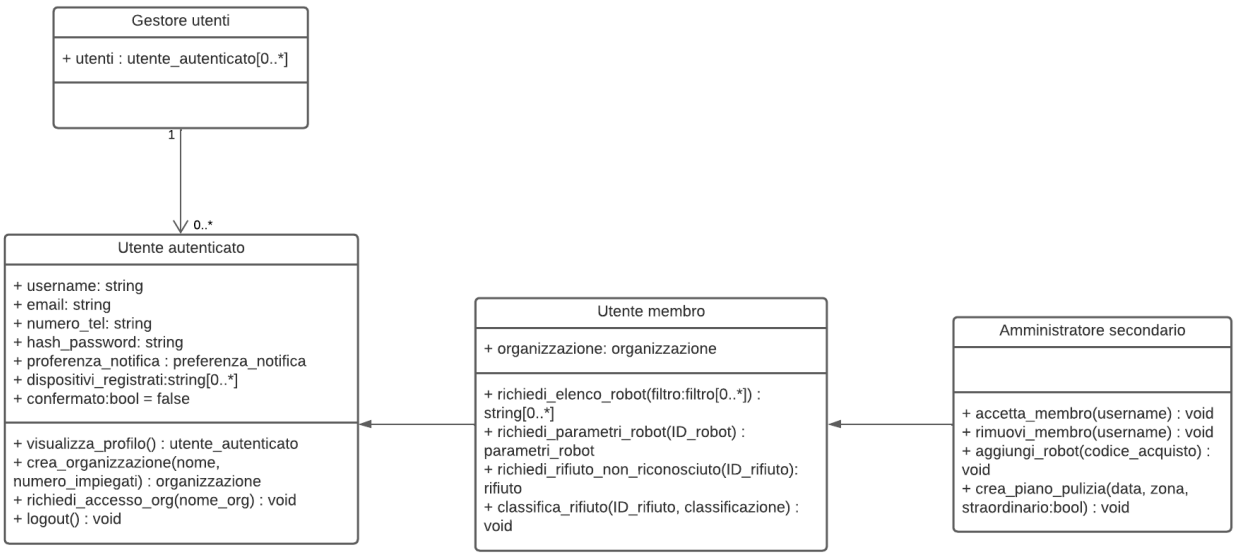
Classificazione rifiuto

Dopo che l'utente ha scelto la corretta classificazione per il rifiuto, quest'ultimo verrà aggiornato all'interno della lista
Espresso in OCL come:

```
context  utente_membro::classifica_rifiuto(ID_rifiuto, classificazione)
post:  self.organizzazione.gestore_piani_pulizia.piani_pulizia->select(c |
c.zona.rifiuti_da_smistare->exists(d | d.ID =
ID_rifiuto)).zona.rifiuti_da_smistare->select(d | d.ID = ID_rifiuto).classificazione =
classificazione
```

2.5 Amministratore secondario

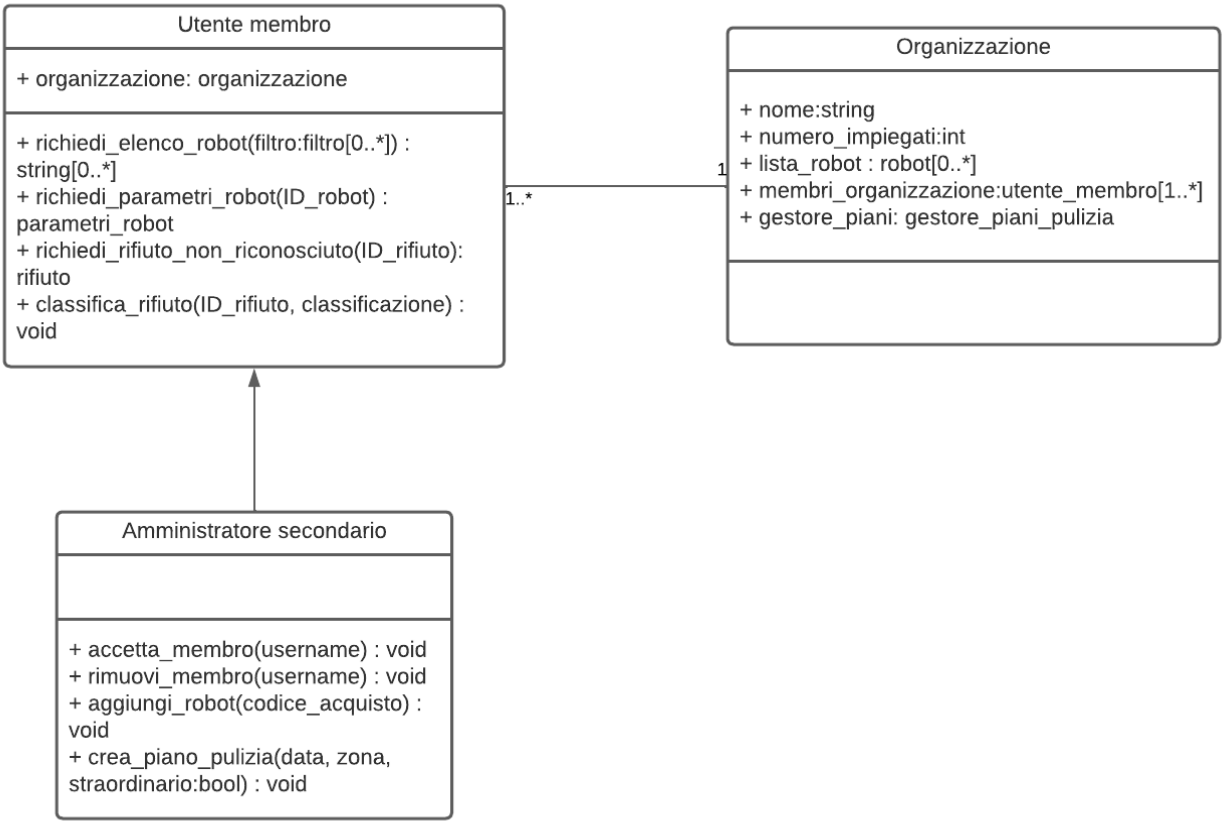
Esistenza membro prima di accettazione



Prima di accettare un nuovo membro, è necessario verificare che esso sia associato ad un account esistente.
Espresso in OCL come:

```
context  amministratore_secondario::accetta_membro(username)
pre:  self.gestore_utenti.utenti->exists(c | c.username = username)
```

Controllo rimozione membro



Per rimuovere un membro da un'organizzazione è necessario che quest'ultimo appartenga all'organizzazione.
Espresso in OCL come:

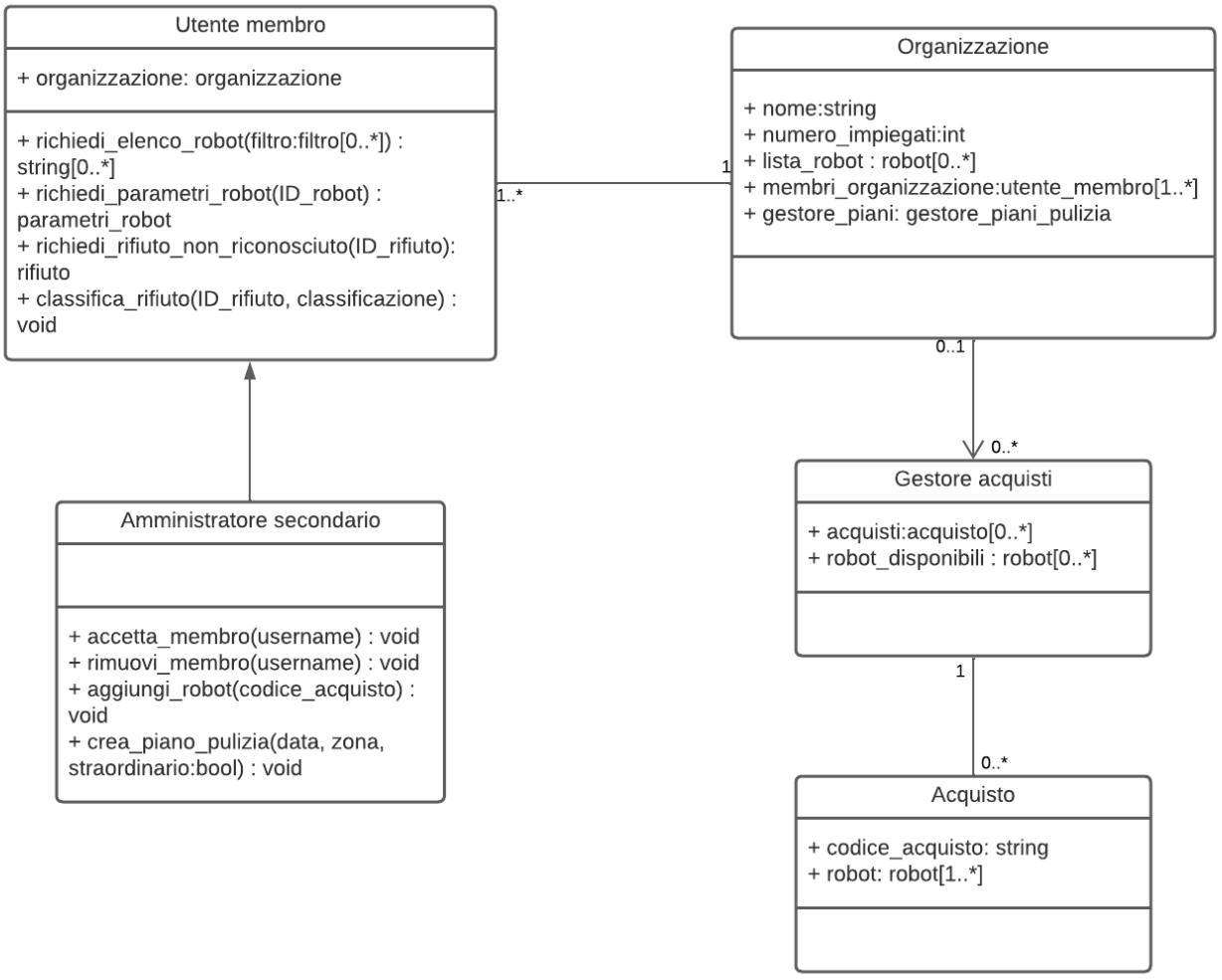
```
context amministratore_secondario::rimuovi_membro(username)
pre: self.organizzazione.membri_organizzazione->exists(c | c.username = username)
```

Rimozione membro

Una volta rimosso un membro, quest'ultimo non farà più parte dell'organizzazione.
Espresso in OCL come:

```
context  amministratore_secondario::rimuovi_membro(username)
post:  !self.organizzazione.membri_organizzazione->exists(c | c.username = username)
```

Validità codice acquisto



Per aggiungere uno o più robot all'organizzazione è necessario inserire un codice d'acquisto valido, ossia un codice contenuto all'interno di un acquisto. Espresso in OCL come:

```
context  amministratore_secondario::aggiungi_robot(codice_acquisto)
pre:  self.organizzazione.gestore_acquisti.acquisti->exists(c | c.codice_acquisto
= codice_acquisto)
```

Rimozione acquisto

Dopo aver inserito un codice d'acquisto corretto, l'acquisto relativo a tale codice verrà rimosso.
Espresso in OCL come:

```
context  amministratore_secondario::aggiungi_robot(codice_acquisto)
post:  !self.organizzazione.gestore_acquisti.acquisti->exists(c | c.codice_acquisto
= codice_acquisto)
```

Validità piano di pulizia

Per creare un nuovo piano di pulizia valido, è necessario inserire una data antecedente a quella attuale.
Espresso in OCL come:

```
context  amministratore_secondario::crea_piano_pulizia(data, zona,
straordinario:bool)
pre:  data > data_attuale
```

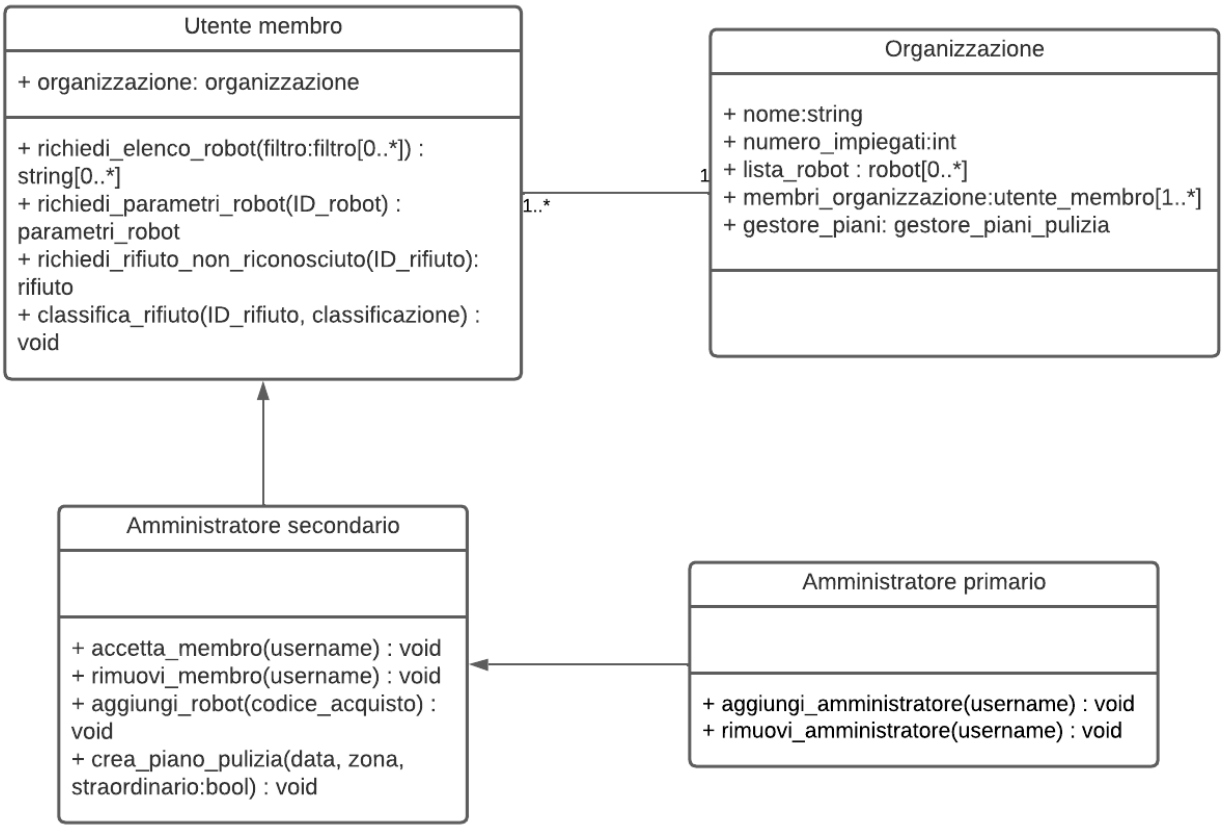
Creazione piano di pulizia

Una volta creato, un piano di pulizia viene memorizzato all'interno del Gestore piani di pulizia.
Espresso in OCL come:

```
context  amministratore_secondario::crea_piano_pulizia(data, zona,
straordinario:bool)
post:  self.organizzazione.gestore_piani_pulizia.piani_pulizia->exists(c | c.data =
data AND c.zona = zona AND c.straordinario = straordinario)
```

2.6 Amministratore primario

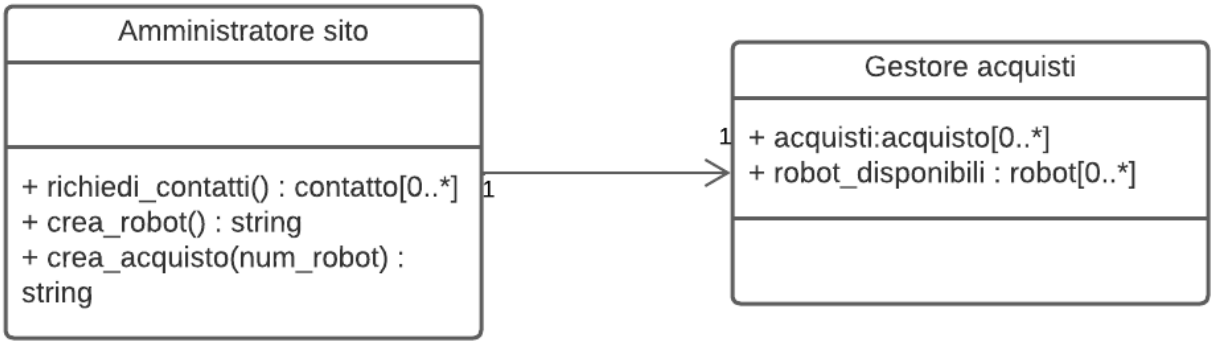
Fornire privilegi di amministratore



Per rendere un utente amministratore di un'organizzazione è necessario che tale utente faccia parte dell'organizzazione.
Espresso in OCL come:

```
context  amministratore_primario::aggiungi_amministratore(username)
pre:  self.organizzazione.membri_organizzazione->exists(c | c.username =
username)
```

2.7 Amministratore del sito



L'amministratore del sito non può creare acquisti contenenti più robot di quelli attualmente disponibili.
Espresso in OCL come:

```
context  amministratore_sito::crea_acquisto(num_robot)
pre:  num_robot < self.gestore_acquisti.robot_disponibili->size()
```