Ardeeno WebApp

T41-SE22

D3-Documento di Architettura

v1.1

alessandro.manfucci@studenti.unitn.it enrico.cescato@studenti.unitn.it
m.sottocornola-1@studenti.unitn.it

29-12-2022

Indice

Abstract

Questo documento, a partire dalle componenti precedentemente definite, descrive l'architettura in classi del sistema da realizzare. L'obiettivo è fornire al team di sviluppo una seconda – e più dettagliata – architettura del software da sviluppare. Si utilizza il diagramma delle classi UML e il linguaggio OCL (*Object Constraint Language*).

1 Diagramma delle classi

Questo capitolo presenta il diagramma delle classi UML del sistema. In particolare si descrivono le singole classi nel loro scopo e contesto, raggruppandole secondo una certa correlazione logica. Le classi descritte rappresentano un'astrazione del codice che sarà eseguito sia lato client, che lato server – omettendo tuttavia la logica di presentazione.

A partire dal diagramma dei componenti si crea per ogni componente almeno una classe; dalle interfacce si derivano gli attributi e i metodi delle classi. Infine, si identificano le relazioni.

Per quanto riguarda le classi, useremo gli stereotipi:

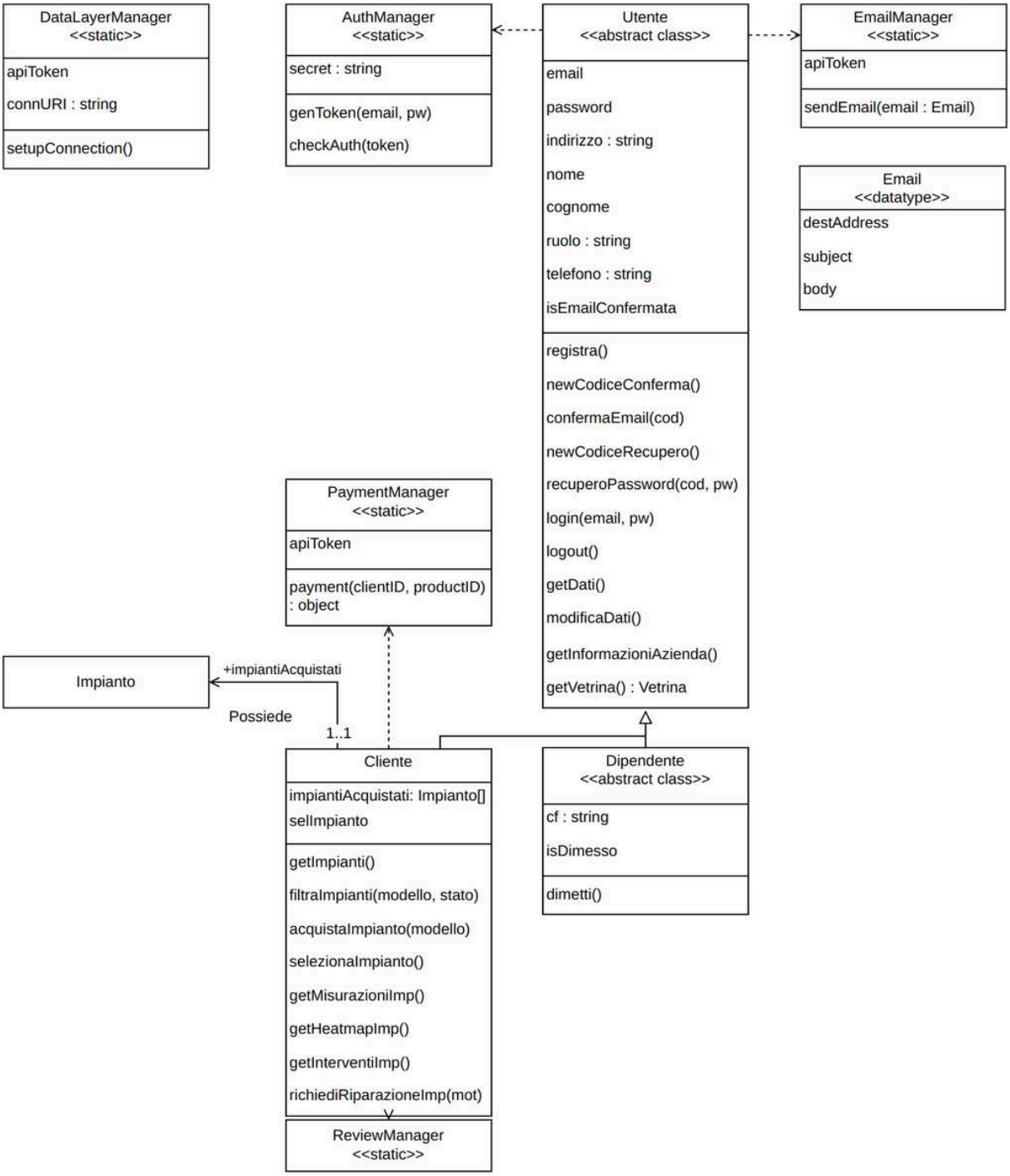
- `<<static>>`: per classi accessibili senza essere instanziate, spesso da implementare solo lato server
- `<<abstract class>>`: per classi non instanziabili ma con metodi concreti
- `<<datatype>>`: per classi senza metodi (impliciti o espliciti)
- `<<enum>>`: per definire nuovi tipi con un dominio limitato agli elementi elencati

Per quanto riguarda gli attributi, si specifica il tipo solo quando vi è ambiguità o si vuole porre un ulteriore vincolo; per quanto riguarda i metodi, per lo stesso principio, ometteremo spesso valore di ritorno, argomenti e tipo degli argomenti. Ometteremo quindi anche la visibilità di metodi e attributi. Inoltre, si considerano impliciti i metodi getter/setter sugli attributi di ogni classe – verranno resi espliciti per dare enfasi al dato attributo. Con il tipo `object` si intende un generico oggetto di cui non ci interessa al momento conoscere la classe.

Per quanto riguarda le relazioni si specifica nome, molteplicità (con notazione *look-ahead*), ruolo e navigabilità solo quando è rilevante. Indicheremo la relazione di dipendenza solo quando non è evidente dal tipo degli attributi, dai parametri dei metodi o dalle altre relazioni – oppure per dare enfasi alla dipendenza.

Diamo il postfisso `Manager` a tutte quelle classi che gestiscono le interazioni con i sistemi esterni.

1.1 Classi relative all'Utente



Dal comp. "Gestione Autenticazione" si definisce la classe `DataLayerManager`, che gestisce le interazioni con il DataLayer. Comprende molti metodi volutamente lasciati impliciti, per non porre vincoli non necessari al team di sviluppo. Certamente vi sarà un metodo `setupConnection()` per stabilire la connessione con il DataLayer tramite una `connUri`. La maggior parte delle classi utilizza i suoi metodi – ovvero ne è dipendente (ma queste relazioni sono omesse dal diagramma).

Dal comp. "Gestione Autenticazione" si definisce la classe `AuthManager`, e dalle due interfacce fornite si hanno i corrispettivi metodi. Inoltre, vi è l'attributo `secret`, utilizzato in `genToken(email, pw)` per avere la garanzia di autenticità del token. L'`Utente` invoca `AuthManager.genToken(email, pw)` nel metodo `login()`, ed invoca `AuthManager.checkAuth(token)` nei metodi `getDati()`, `modificaDati()` – dunque ne è dipendente. Anche le sottoclassi di `Utente` ne sono dipendenti – in generale tutti i metodi

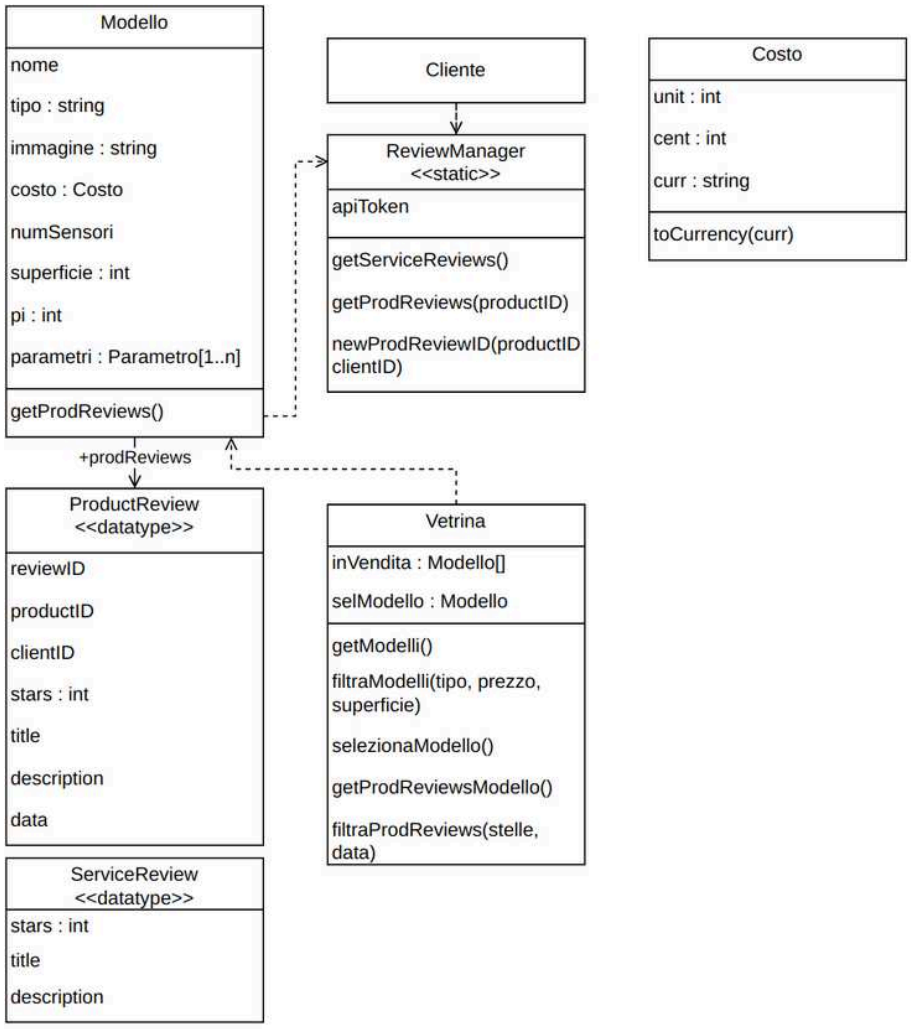
che accedono ai dati dello specifico Utente invocano `AuthManager.checkAuth(token)`.

Dal comp. "Gestione Gmail" si definisce la classe `EmailManager` con il metodo `sendEmail(email)`. L'Utente e le sue sottoclassi ne sono dipendenti. Come anticipato nel D2, si utilizza il «datatype» `Email` come formato standard utilizzato all'interno del nostro sistema.

Dal comp. "Gestione PayPal" si definisce la classe `PaymentManager`. Vi è un metodo `payment(clientID, productID) : object` di cui sappiamo che vi è un `object` come valore di ritorno, ma non la specifica classe.

Il `Cliente` dipende dalla classe `ReviewManager` poiché la utilizza per ottenere il codice product review al momento di acquisto dell'impianto. Il metodo `selezionaImpianto()` fornisce la Dashboard dell'impianto come definita in RF11.1, da cui si accede alle funzioni espresse dai metodi successivi.

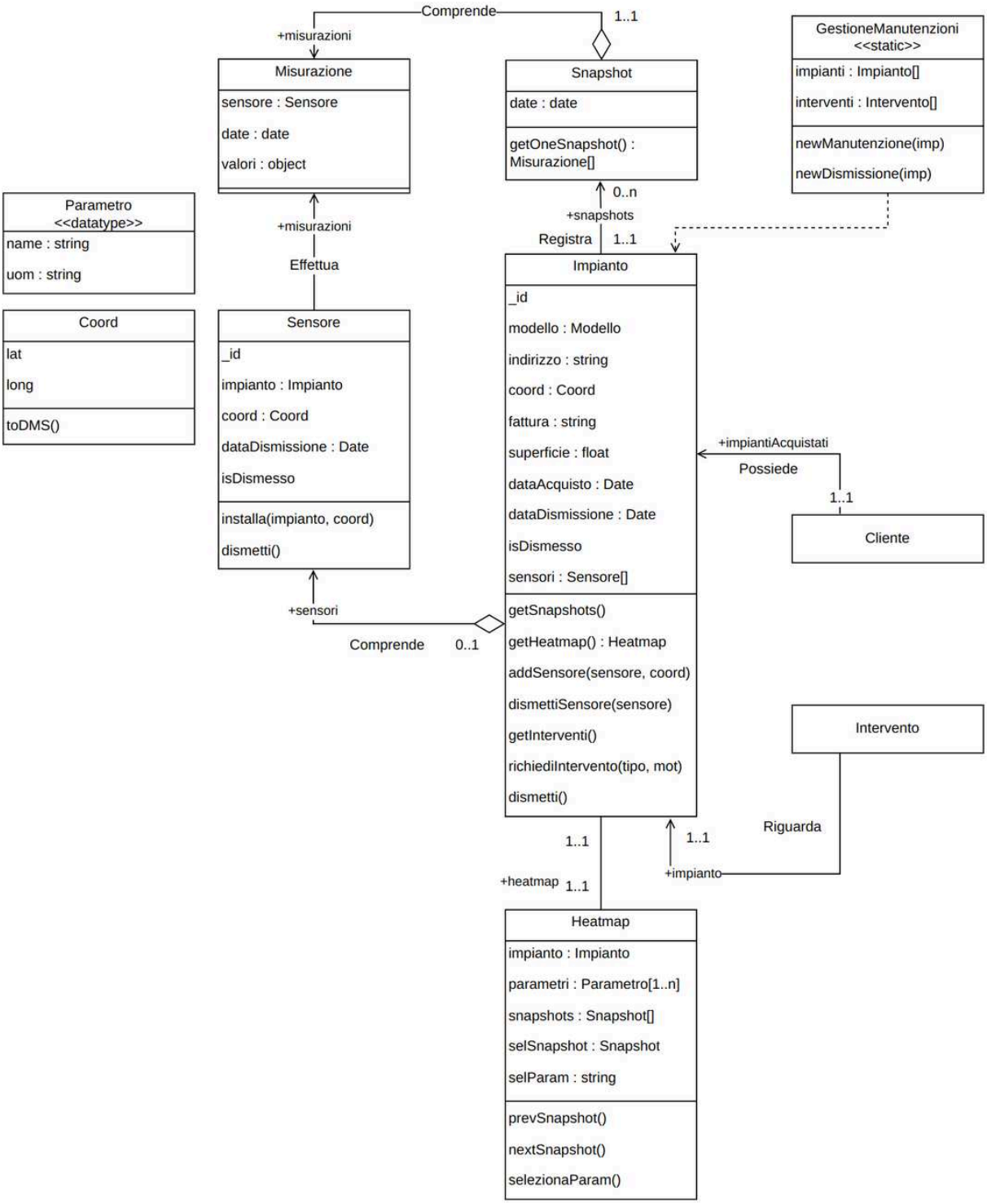
1.2 Classi relative alla Vetrina



Dal comp. "Vetrina" si definiscono le classi `Vetrina`, `Modello`, `Costo`. Si osservi che il `Modello` ha un attributo `parametri : Parametro [1..n]` che specifica i parametri misurati dal modello di impianto e un attributo `costo : Costo`, dove `Costo` è una classe con ulteriori metodi.

Dal comp. "Gestione TrustPilot" si definiscono le classi `ReviewManager`, `ProductReview`, `ServiceReview`. Il metodo `newProdReviewID(productID, clientID)` si interfaccia con TrustPilot e ritorna il codice product review; il `productID` è necessario per associare la product review al modello di impianto che è stato acquistato. Utilizzando questo processo ogni product review è associata ad un acquisto realmente effettuato.

1.3 Classi relative all'Impianto



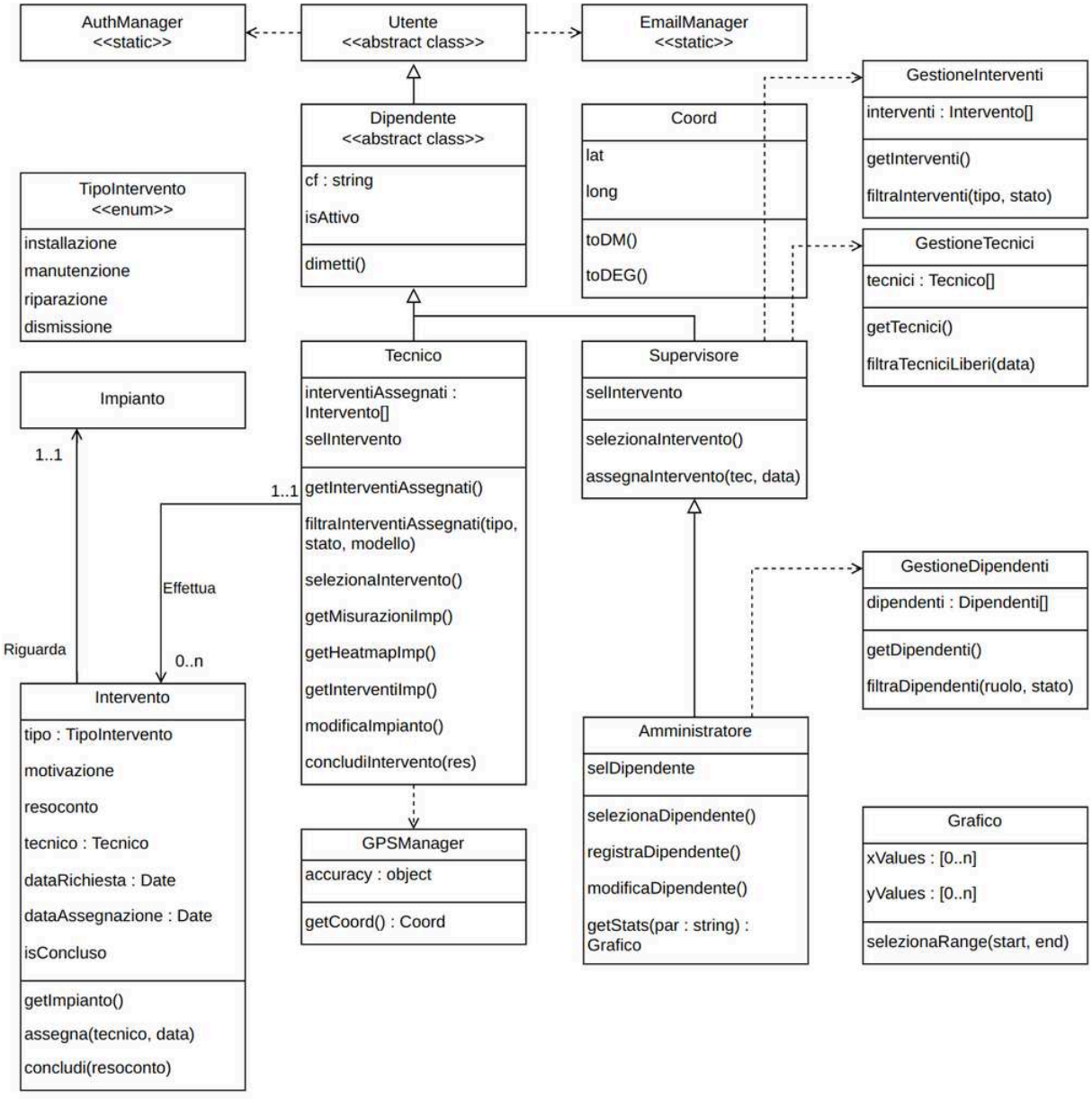
Dal comp. "Gestione Impianto Acquistato" si definiscono le classi `Impianto`, `Snapshot`, `Misurazione`, `Sensore`, `Heatmap`, `GestioneManutenzione`. Queste classi rappresentano le risorse principali gestite dal sistema.

La classe `Impianto` ha un attributo `coord : Coord` che **non** è inserito dal Cliente al momento dell'acquisto e **non** rappresenta le coordinate geografiche dell'`indirizzo`. Bensì è un campo calcolato a partire dalle coordinate dei `sensori : Sensore[]` – rappresenta il centro dell'impianto.

La classe `Sensore` necessita di `dataDismissione` poiché la possibile alternativa – che prevede la rimozione del Sensore dal DataLayer – non permette di visualizzare sulla `Heatmap` le misurazioni di un `Sensore` dismesso (si osservi che non si avrebbero più le coordinate della misurazione).

La classe `GestioneManutenzioni` inserisce nel DataLayer interventi di Manutenzione su ogni `Impianto` con periodicità annuale (come richiesto dai requisiti del sistema) ed esattamente quando sono trascorsi 5 anni dalla data di acquisto dell'`Impianto` richiede un intervento di Dismissione. È quindi dipendente da `Impianto.richiediIntervento(tipo, mot)`.

1.4 Classi relative ai Dipendenti



Queste classi rappresentano la parte operativa del sistema. Dal comp. "Gestione Supervisore" si definiscono le classi `Supervisore`, `GestioneInterventi`, `GestioneTecnici`. Anche se omesso dal diagramma, il `Supervisore` è dipendente dai metodi di `Intervento`. Le classi di `Gestione...` hanno lo scopo di alleggerire il `Supervisore` dalla logica di filtraggio e dalle interazioni con il DataLayer – inoltre, è bene vedere gli `Interventi` come distaccati dal `Supervisore` e maggiormente legati al `Tecnico`.

Dal comp. "Gestione Tecnico" si definisce la classe `Tecnico`, `Intervento`, `TipoIntervento`. La classe `Tecnico` è dipendente dai metodi di `Intervento`.

Dal comp. “Gestione GPS” si definiscono le classi `GPSManager`, `Coord`. Come anticipato nel D2, la classe `Coord` è uno standard semplificato interno per la rappresentazione delle coordinate.

Dal comp. “Gestione Amministratore” si definiscono le classi `Amministratore`, `GestioneDipendenti`, `Grafico`

1.5 Diagramma delle classi complessivo

Alla fine del documento si trova il diagramma leggibile.

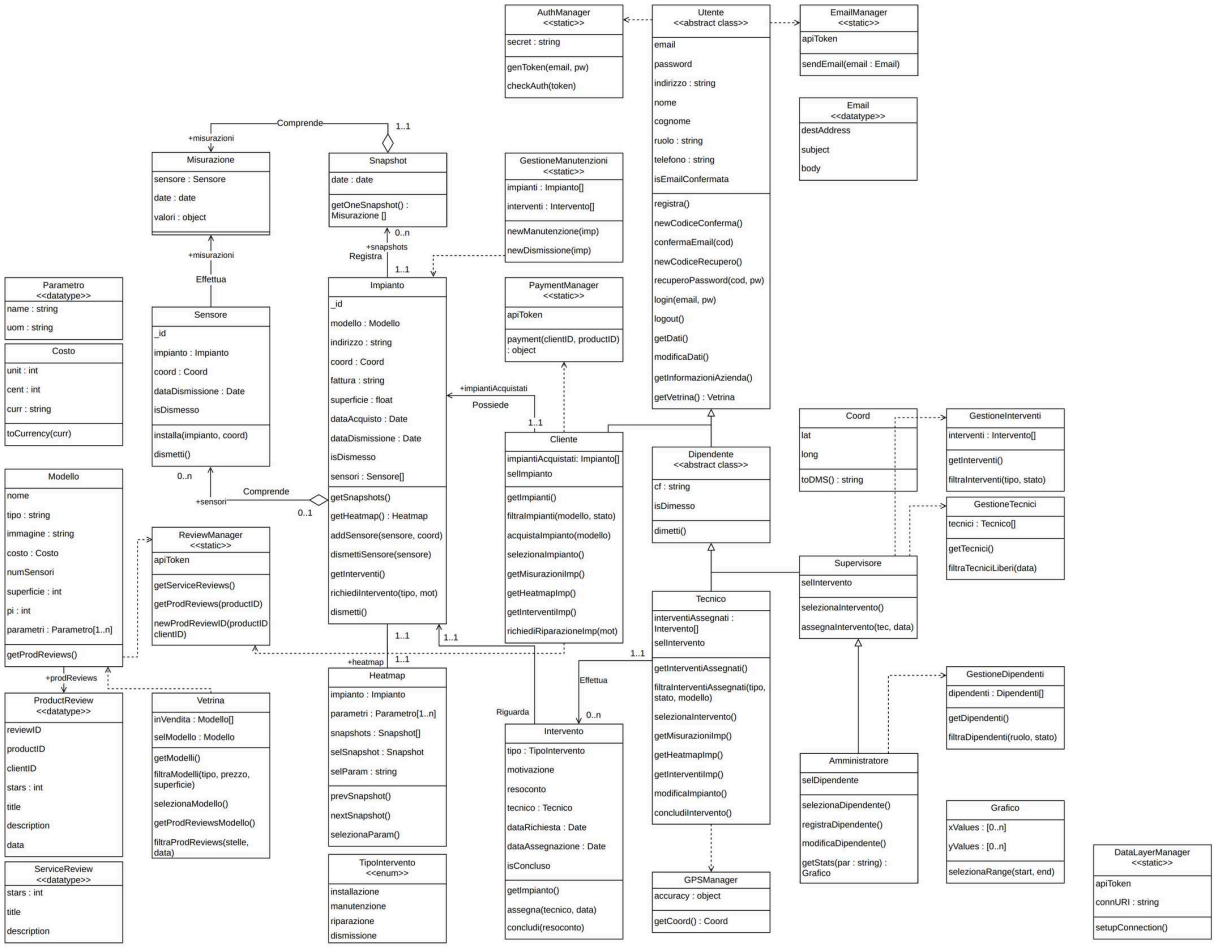


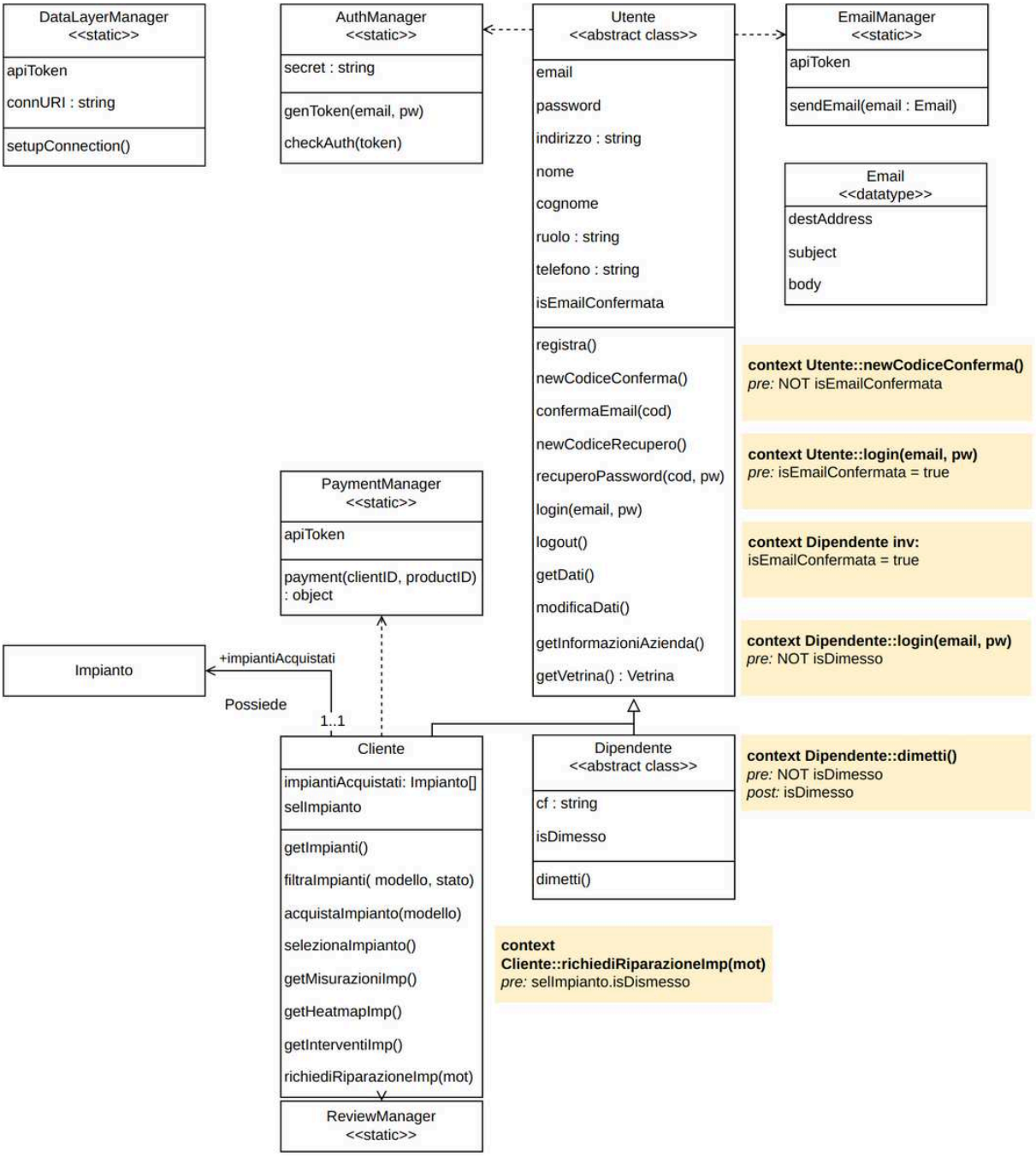
Figura 1: Anteprima diagramma delle classi complessivo

2 Codice in OCL

Questo capitolo presenta particolari proprietà sugli attributi e sulle operazioni delle varie classi (invarianti, pre-condizioni e post-condizioni). Come nel capitolo precedente, le classi sono raggruppate secondo una certa correlazione logica.
Si utilizza il linguaggio OCL 2.4, che estende lo standard UML.

Per quanto riguarda le espressione booleane OCL, per attributi di tipo boolean ometteremo spesso `someBoolAttr = true` scrivendo semplicemente `someBoolAttr`. Si aggiunge l'operazione globale `now()` per indicare il valore `Date` che indica la data corrente e `tomorrow()` per indicare il valore `Date` che indica il giorno successivo a `now()`. Inoltre estenderemo l'OCL con l'operatore binario IF-AND-ONLY-IF definito come `(a IIF b) = (a IMPLIES b) AND (b IMPLIES a)`.

2.1 Classi relative all'Utente



2.1.1 Utente

L'`Utente` può richiedere un codice di conferma solo se l'email non è confermata. Questo vincolo è presente poiché `newCodiceConferma()` invia un'email tramite il sistema esterno Gmail – con un certo costo.

```
context Utente::newCodiceConferma()
pre: NOT isEmailConfermata
```

L'`Utente` può effettuare il login solo se l'email è confermata.

```
context Utente::login(email, pw)
pre: isEmailConfermata = true
```

2.1.2 Dipendente

Un `Dipendente` ha l'email confermata per default, senza necessità di effettuare il processo di conferma. Questo vincolo è accennato nello UseCase Registrazione dipendente.

```
context Dipendente inv:
isEmailConfermata = true
```

Un `Dipendente` non può effettuare il login se è dimesso. Si osservi che per la gerarchia di sottoclasse, per il Dipendente valgono anche le `context Utente::login()` ma sono sempre soddisfatte.

```
context Dipendente::login(email, pw)
pre: NOT isDimesso
```

Il metodo `dimetti()` opera sull'attributo `isDimesso`

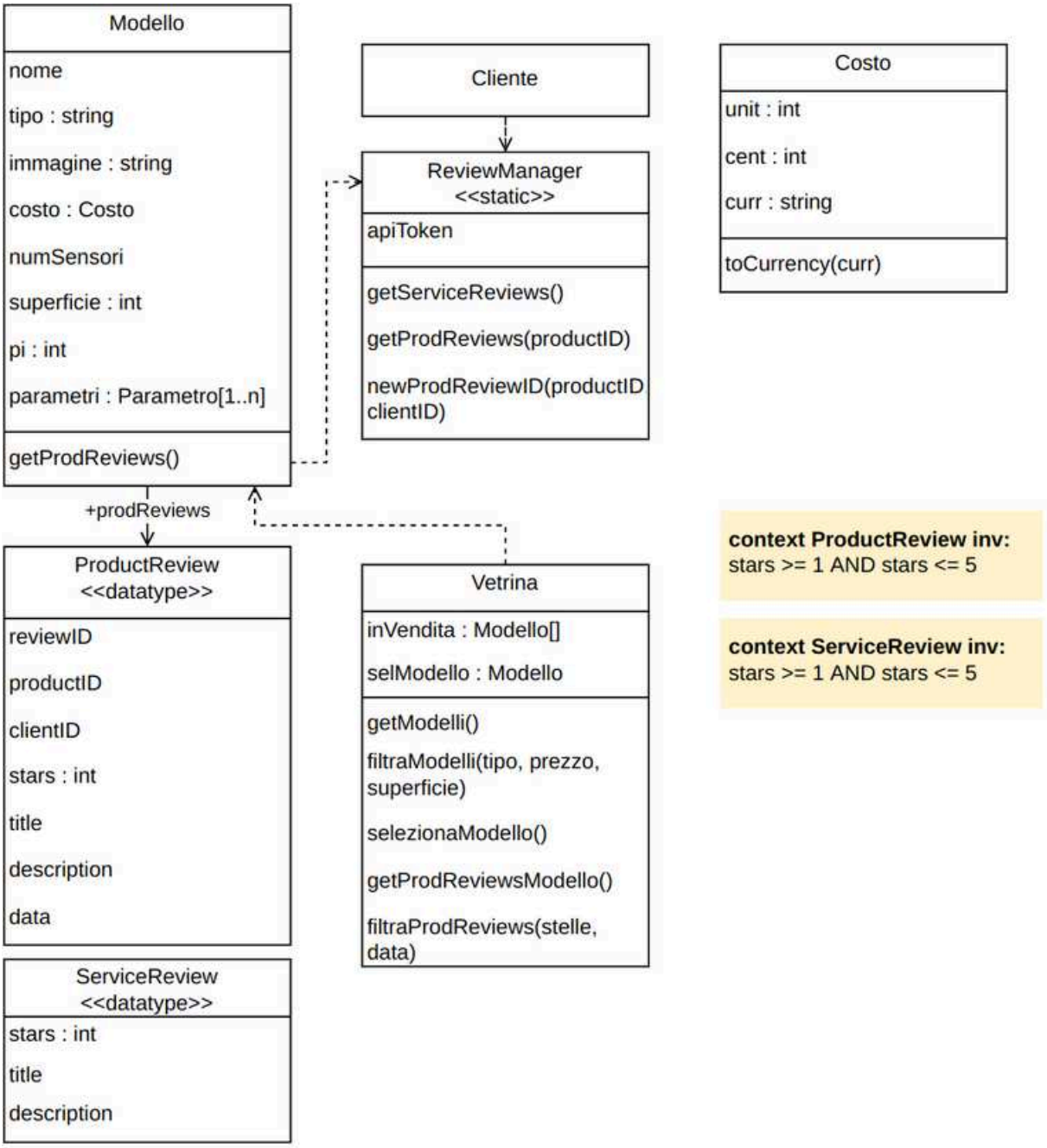
```
context Dipendente::dimetti()
pre: NOT isDimesso
post: isDimesso
```

2.1.3 Cliente

Il metodo `richiediRiparazioneImp(mot)` può essere invocato solo su un impianto non dismesso

```
context Cliente::richiediRiparazioneImp(mot)
pre: NOT sel.Impianto.isDismesso
```

2.2 Classi relative alla Vetrina



2.2.1 ProductReview

`stars` è compreso tra 1 e 5

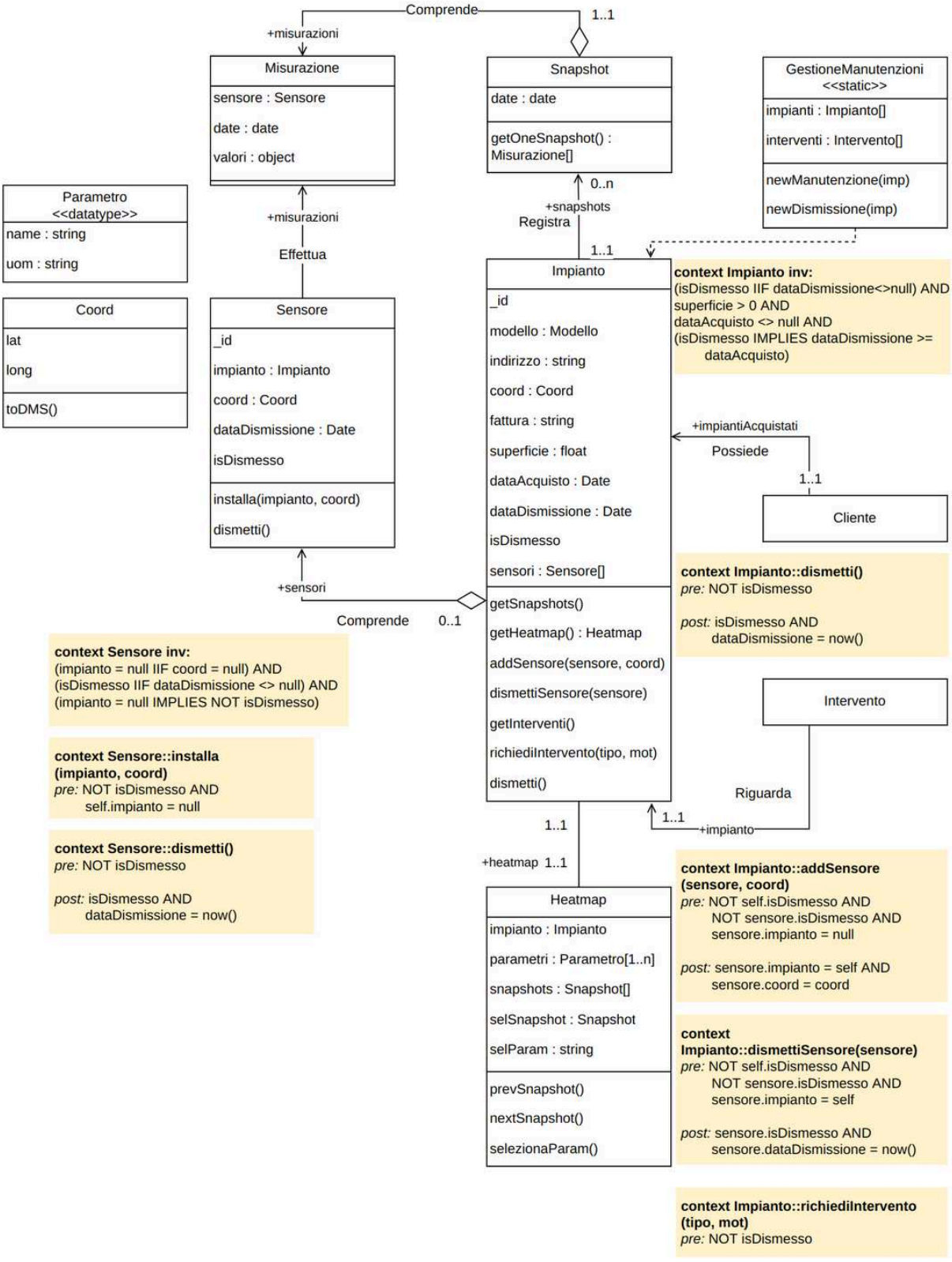
```
context ProductReview inv:
stars >= 1 AND stars <= 5
```

2.2.2 ServiceReview

`stars` è compreso tra 1 e 5

```
context ServiceReview inv:
stars >= 1 AND stars <= 5
```

2.3 Classi relative all'Impianto



2.3.1 Impianto

`isDismesso` è un attributo booleano ridondante; `dataAcquisto` è sempre impostato; se l'impianto è dismesso, `dataDismissione` è impostato ed è più recente di `dataAcquisto`. Si comprendono i casi di eccezione in cui l'impianto è dismesso prima dei 5 anni di servizio stabiliti.

```
context Impianto inv:
(isDismesso IIF dataDismissione = null) AND superficie > 0 AND
dataAcquisto <> null AND
(isDismesso IMPLIES dataDismissione >=  dataAcquisto)
```

`dismetti()` imposta `isDismesso`, `dataDismissione`

```
context Impianto::dismetti()
pre: NOT isDismesso

post: isDismesso AND
    dataDismissione = now()
```

Il metodo `addSensore(sensore, coord)` imposta `sensore`, `coord` e opera su un impianto non dismesso e su un sensore non dismesso e non compreso in alcun impianto.

```
context Impianto::addSensore(sensore, coord)
pre: NOT self.isDismesso AND
    NOT sensore.isDismesso AND
    sensore.impianto = null

post: sensore.impianto = self AND
    sensore.coord = coord
```

Il metodo `dismettiSensore(sensore)` imposta `sensore.isDismesso`, `sensore.dataDismissione` ed opera su un impianto non dismesso e su un sensore compreso nel dato impianto e non dismesso

```
context Impianto::dismettiSensore(sensore)
pre: NOT self.isDismesso AND
    NOT sensore.isDismesso AND
    sensore.impianto = self

post: sensore.isDismesso AND
    sensore.dataDismissione = now()
```

Non è possibile richiedere un intervento su un impianto dismesso.

```
context Impianto::richiediIntervento(tipo, mot)
pre: NOT isDismesso
```

2.3.2 Sensore

Un `Sensore` è installato sempre a delle coordinate; `isDismesso` è un attributo booleano ridondante; un `Sensore` non può essere dismesso se non è mai installato.

```
context Sensore inv:
(impianto = null IIF coord = null) AND
(isDismesso IIF dataDismissione <> null) AND
(impianto = null IMPLIES NOT isDismesso)
```

Il metodo `installa(impianto, coord)` opera su un `Sensore` non dismesso e non installato.

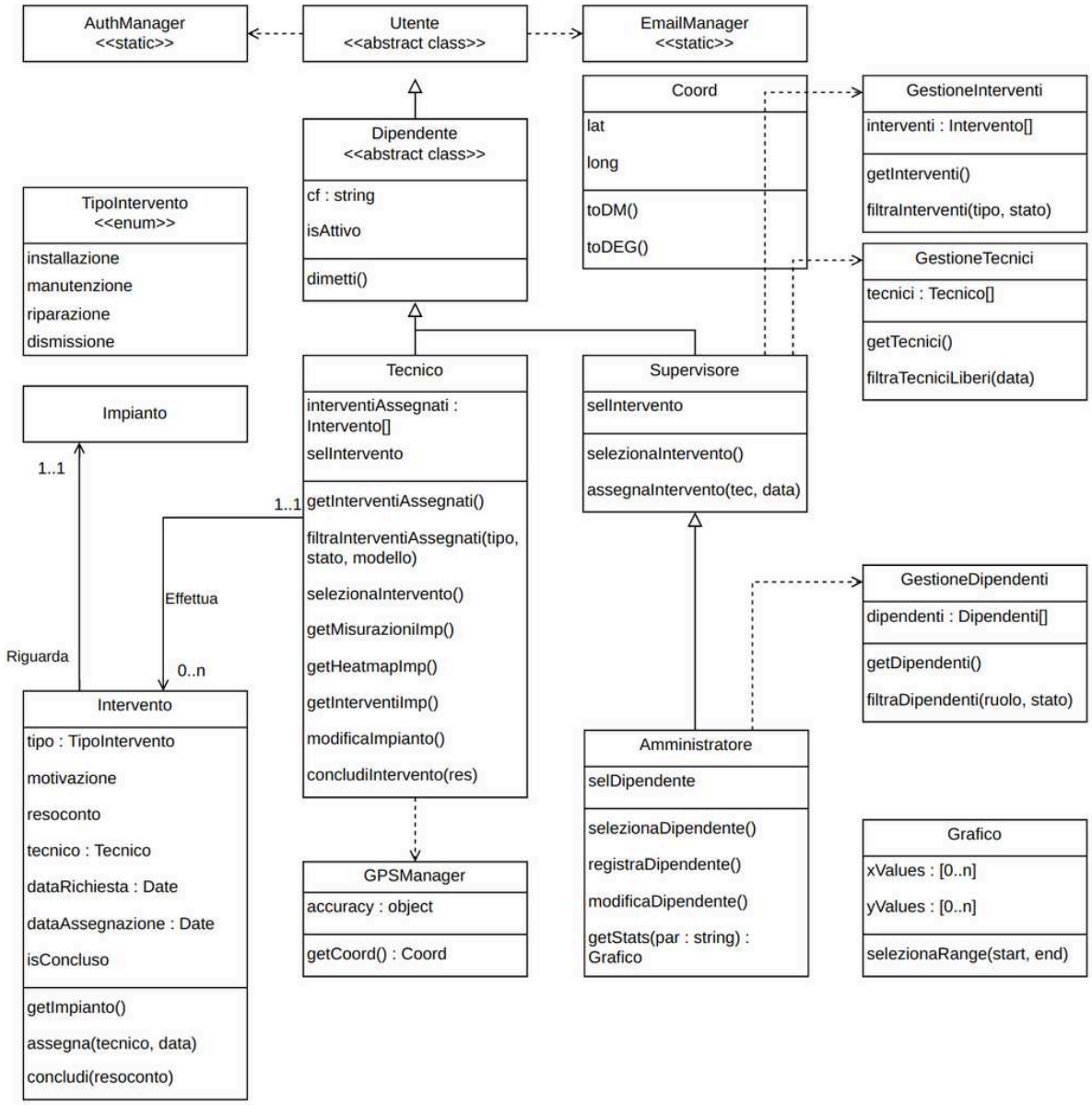
```
context Sensore::installa(impianto, coord)
pre: NOT isDismesso AND
    self.impianto = null
```

Il metodo `dismetti()` imposta `isDismesso`, `dataDismissione` ed opera su un `Sensore` non dismesso.

```
context Sensore::dismetti()
pre: NOT isDismesso

post: isDismesso AND
    dataDismissione = now()
```

2.4 Classi relative ai Dipendenti



context Intervento inv:
dataRichiesta <> null AND
(motivazione <> null IIF tipo = riparazione)
(tecnico = null IIF dataAssegnazione = null) AND
(isConcluso IIF resoconto <> null) AND
(dataAssegnazione <> null IMPLIES
dataAssegnazione >= dataRichiesta)

context Intervento::assegna(tecnico, data)
pre: self.tecnico = null AND
self.dataAssegnazione = null

post: self.tecnico = tecnico AND
self.dataAssegnazione = data

context Intervento::concludi(resoconto)
pre: NOT isConcluso

post: self.resoconto = resoconto AND
isConcluso

context Tecnico inv:
interventiAssegnati->forAll(i | i.tecnico = self)

context Tecnico::concludiIntervento(res)
pre: NOT selIntervento.isConcluso

post: selIntervento.resoconto = res AND
selIntervento.isConcluso

context Supervisore::assegnaIntervento (tec, data)
pre: selIntervento.tecnico = null AND
data >= tomorrow() AND
tec.interventiAssegnati->forAll(i |
i.dataAssegnazione <> data)

post: selIntervento.tecnico = tec AND
selIntervento.dataAssegnazione = now()

context Tecnico::getMisurazioniImp()
pre: NOT selIntervento.isConcluso

context Tecnico::getHeatmapImp()
pre: NOT selIntervento.isConcluso

context Tecnico::getInterventiImp()
pre: NOT selIntervento.isConcluso

context Tecnico::modificaImpianto()
pre: NOT selIntervento.isConcluso

2.4.1 Intervento

La `motivazione` è impostata solo per gli interventi di riparazione; un `tecnico` è sempre assegnato in una `dataAssegnazione`; `isConcluso` è un attributo ridondante; se l'intervento è assegnato, la `dataAssegnazione` è maggiore della `dataRichiesta`.

```
context Intervento inv:
dataRichiesta <> null AND
(motivazione <> null IIF tipo = riparazione)
(tecnico = null IIF dataAssegnazione = null) AND
(isConcluso IIF resoconto <> null) AND
(dataAssegnazione <> null IMPLIES dataAssegnazione >= dataRichiesta)
```

Il metodo imposta `tecnico`, `data` e opera su un `Intervento` non ancora assegnato.

```
context Intervento::assegna(tecnico, data)
pre: self.tecnico = null AND
    self.dataAssegnazione = null

post: self.tecnico = tecnico AND
    self.dataAssegnazione = data
```

Il metodo imposta `isConcluso`, `resoconto` e opera su un `Intervento` non concluso

```
context Intervento::concludi(resoconto)
pre: NOT isConcluso

post: self.resoconto = resoconto AND
    isConcluso
```

2.4.2 Tecnico

Definizione di intervento assegnato a un tecnico.

```
context Tecnico inv:
interventiAssegnati->forAll( i | i.tecnico = self)
```

I metodi `getMisurazioniImp()`, `getHeatmapImp()`, `getInterventiImp()`, `modificaImpianto()`, `modificaImpianto()` possono essere invocati solo se `selIntervento` non è concluso.

```
context Tecnico::getMisurazioniImp()
pre: NOT selIntervento.isConcluso
```

```
context Tecnico::getHeatmapImp()
pre: NOT selIntervento.isConcluso
```

```
context Tecnico::getInterventiImp()
pre: NOT selIntervento.isConcluso
```

```
context Tecnico::modificaImpianto()
pre: NOT selIntervento.isConcluso
```

```
context Tecnico::concludiIntervento(res)
pre: NOT selIntervento.isConcluso

post: selIntervento.resoconto = res AND
    selIntervento.isConcluso
```

Il metodo `concludiIntervento(res)` imposta `selIntervento.resoconto` ed `selIntervento.isConcluso`.

2.4.3 Supervisore

Il `Supervisore` può assegnare un `Intervento` a un `Tecnico` solo se l'`Intervento` non è concluso e il `Tecnico` è libero nella `Data`; la `Data` non può essere il giorno corrente.

```
context Supervisore::assegnaIntervento(tec, data)
pre: selIntervento.tecnico = null AND
  data >= tomorrow() AND
    tec.interventiAssegnati->forAll( i |
      i.dataAssegnazione <> data)

post: selIntervento.tecnico = tec AND
    selIntervento.dataAssegnazione = data
```

2.5 Diagramma delle classi complessivo con codice OCL

Alla fine del documento si trova il diagramma leggibile.

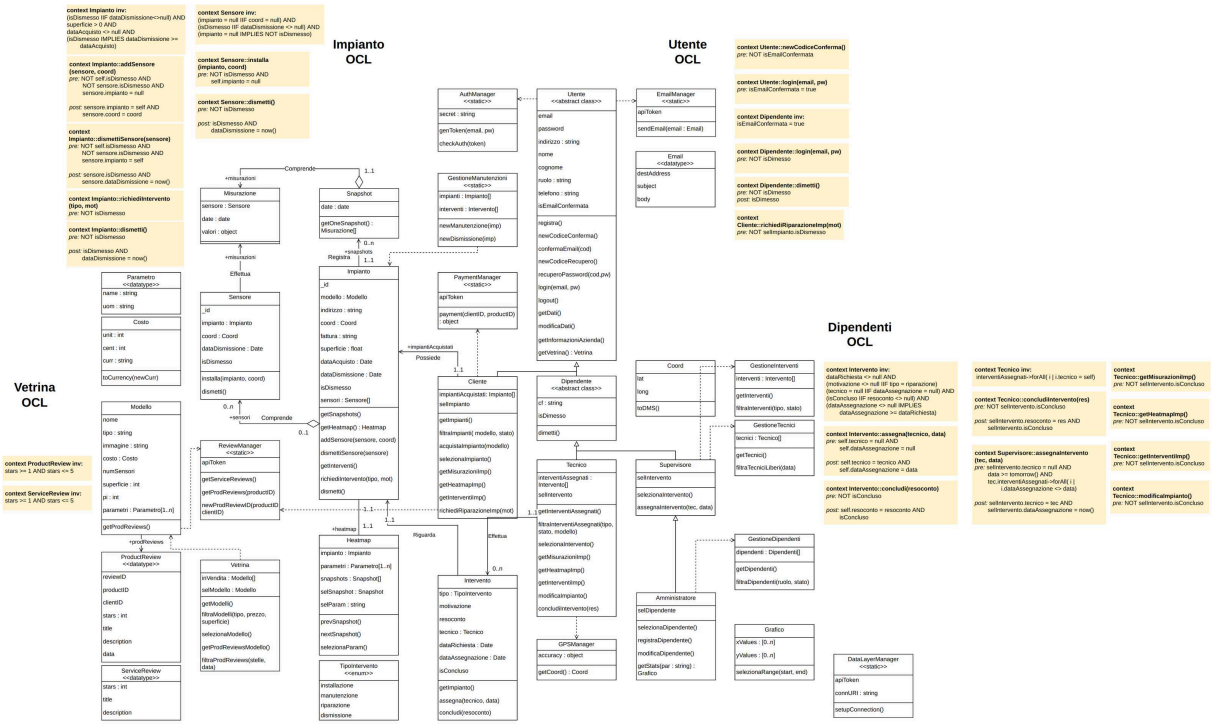
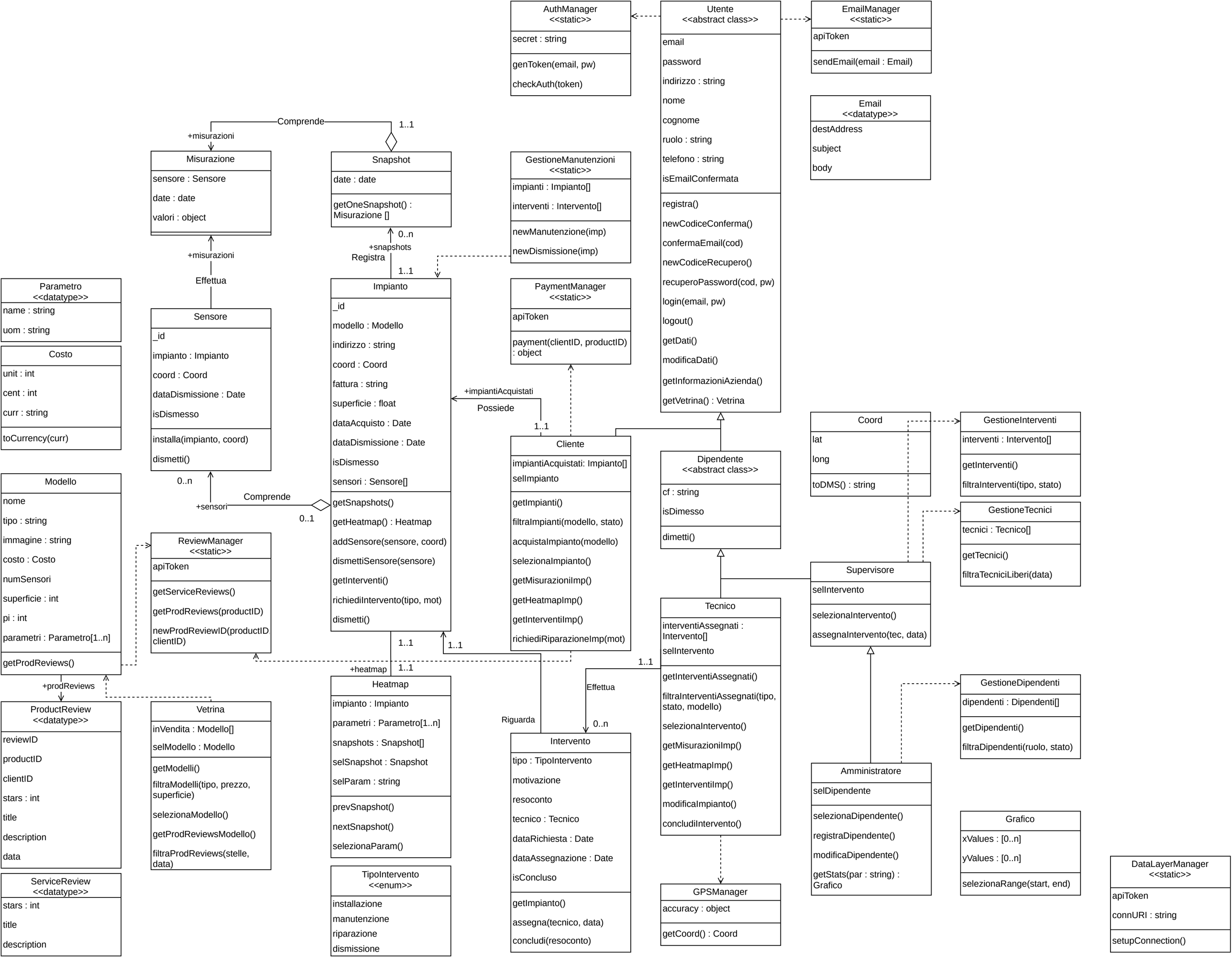


Figura 2: Anteprima diagramma delle classi complessivo con codice OCL


AuthManager
<<static>>
secret : string
genToken(email, pw)
checkAuth(token)
Utente
<<abstract class>>
email
password
indirizzo : string
nome
cognome
ruolo : string
telefono : string
isEmailConfermata
registra()
newCodiceConferma()
confermaEmail(cod)
newCodiceRecupero()
recuperoPassword(cod, pw)
login(email, pw)
logout()
getDati()
modificaDati()
getInformazioniAzienda()
getVetrina() : Vetrina
EmailManager
<<static>>
apiToken
sendEmail(email : Email)
Email
<<datatype>>
destAddress
subject
body
Comprende
1..1
+misurazioni
Misurazione
sensore : Sensore
date : date
valori : object
Snapshot
date : date
getOneSnapshot() :
Misurazione []
GestioneManutenzioni
<<static>>
impianti : Impianto[]
interventi : Intervento[]
newManutenzione(imp)
newDismissione(imp)
0..n
+snapshots
Registra
1..1
+misurazioni
Effettua
Parametro
<<datatype>>
name : string
uom : string
Costo
unit : int
cent : int
curr : string
toCurrency(curr)
Sensore
_id
impianto : Impianto
coord : Coord
dataDismissione : Date
isDismesso
installa(impianto, coord)
dismetti()
Impianto
_id
modello : Modello
indirizzo : string
coord : Coord
fattura : string
superficie : float
dataAcquisto : Date
dataDismissione : Date
isDismesso
sensori : Sensore[]
getSnapshots()
getHeatmap() : Heatmap
addSensore(sensore, coord)
dismettiSensore(sensore)
getInterventi()
richiediIntervento(tipo, mot)
dismetti()
PaymentManager
<<static>>
apiToken
payment(clientID, productID)
: object
+impiantiAcquistati
Possiede
1..1
0..n
Comprende
+sensori
0..1
Modello
nome
tipo : string
immagine : string
costo : Costo
numSensori
superficie : int
pi : int
parametri : Parametro[1..n]
getProdReviews()
ReviewManager
<<static>>
apiToken
getServiceReviews()
getProdReviews(productID)
newProdReviewID(productID
clientID)
+heatmap
1..1
1..1
1..1
Heatmap
impianto : Impianto
parametri : Parametro[1..n]
snapshots : Snapshot[]
selSnapshot : Snapshot
selParam : string
prevSnapshot()
nextSnapshot()
selezionaParam()
Cliente
impiantiAcquistati: Impianto[]
selImpianto
getImpianti()
filtraImpianti(modello, stato)
acquistaImpianto(modello)
selezionaImpianto()
getMisurazioniImp()
getHeatmapImp()
getInterventiImp()
richiediRiparazioneImp(mot)
Dipendente
<<abstract class>>
cf : string
isDimesso
dimetti()
Coord
lat
long
toDMS() : string
GestioneInterventi
interventi : Intervento[]
getInterventi()
filtraInterventi(tipo, stato)
GestioneTecnici
tecnici : Tecnico[]
getTecnici()
filtraTecniciLiberi(data)
Supervisore
selIntervento
selezionaIntervento()
assegnaIntervento(tec, data)
Tecnico
interventiAssegnati :
Intervento[]
selIntervento
getInterventiAssegnati()
filtraInterventiAssegnati(tipo,
stato, modello)
selezionaIntervento()
getMisurazioniImp()
getHeatmapImp()
getInterventiImp()
modificaImpianto()
concludiIntervento()
Effettua
Riguarda
0..n
Intervento
tipo : TipoIntervento
motivazione
resoconto
tecnico : Tecnico
dataRichiesta : Date
dataAssegnazione : Date
isConcluso
getImpianto()
assegna(tecnico, data)
concludi(resoconto)
+prodReviews
ProductReview
<<datatype>>
reviewID
productID
clientID
stars : int
title
description
data
Vetrina
inVendita : Modello[]
selModello : Modello
getModelli()
filtraModelli(tipo, prezzo,
superficie)
selezionaModello()
getProdReviewsModello()
filtraProdReviews(stelle,
data)
ServiceReview
<<datatype>>
stars : int
title
description
TipoIntervento
<<enum>>
installazione
manutenzione
riparazione
dismissione
GPSManager
accuracy : object
getCoord() : Coord
Amministratore
selDipendente
selezionaDipendente()
registraDipendente()
modificaDipendente()
getStats(par : string) :
Grafico
GestioneDipendenti
dipendenti : Dipendenti[]
getDipendenti()
filtraDipendenti(ruolo, stato)
Grafico
xValues : [0..n]
yValues : [0..n]
selezionaRange(start, end)
DataLayerManager
<<static>>
apiToken
connURI : string
setupConnection()

context Impianto inv:
(isDismesso IIF dataDismissione<>null) AND
superficie > 0 AND
dataAcquisto <> null AND
(isDismesso IMPLIES dataDismissione >=
dataAcquisto)

context Impianto::addSensore (sensore, coord)
pre: NOT self.isDismesso AND
NOT sensore.isDismesso AND
sensore.impianto = null

post: sensore.impianto = self AND
sensore.coord = coord

context Impianto::dismettiSensore(sensore)
pre: NOT self.isDismesso AND
NOT sensore.isDismesso AND
sensore.impianto = self

post: sensore.isDismesso AND
sensore.dataDismissione = now()

context Impianto::richiediIntervento (tipo, mot)
pre: NOT isDismesso

context Impianto::dismetti()
pre: NOT isDismesso

post: isDismesso AND
dataDismissione = now()

context Sensore inv:
(impianto = null IIF coord = null) AND
(isDismesso IIF dataDismissione <> null) AND
(impianto = null IMPLIES NOT isDismesso)

context Sensore::installa (impianto, coord)
pre: NOT isDismesso AND
self.impianto = null

context Sensore::dismetti()
pre: NOT isDismesso

post: isDismesso AND
dataDismissione = now()

Impianto OCL

Utente OCL

context Utente::newCodiceConferma()
pre: NOT isEmailConfermata

context Utente::login(email, pw)
pre: isEmailConfermata = true

context Dipendente inv:
isEmailConfermata = true

context Dipendente::login(email, pw)
pre: NOT isDimesso

context Dipendente::dimetti()
pre: NOT isDimesso
post: isDimesso

context Cliente::richiediRiparazioneImp(mot)
pre: NOT selImpianto.isDismesso

Vetrina OCL

context ProductReview inv:
stars >= 1 AND stars <= 5

context ServiceReview inv:
stars >= 1 AND stars <= 5

Dipendenti OCL

context Intervento inv:
dataRichiesta <> null AND
(motivazione <> null IIF tipo = riparazione)
(tecnico = null IIF dataAssegnazione = null) AND
(isConcluso IIF resoconto <> null) AND
(dataAssegnazione <> null IMPLIES
dataAssegnazione >= dataRichiesta)

context Intervento::assegna(tecnico, data)
pre: self.tecnico = null AND
self.dataAssegnazione = null

post: self.tecnico = tecnico AND
self.dataAssegnazione = data

context Intervento::concludi(resoconto)
pre: NOT isConcluso

post: self.resoconto = resoconto AND
isConcluso

context Tecnico inv:
interventiAssegnati->forAll(i | i.tecnico = self)

context Tecnico::concludiIntervento(res)
pre: NOT selIntervento.isConcluso

post: selIntervento.resoconto = res AND
selIntervento.isConcluso

context Supervisore::assegnaIntervento (tec, data)
pre: selIntervento.tecnico = null AND
data >= tomorrow() AND
tec.interventiAssegnati->forAll(i |
i.dataAssegnazione <> data)

post: selIntervento.tecnico = tec AND
selIntervento.dataAssegnazione = now()

context Tecnico::getMisurazioniImp()
pre: NOT selIntervento.isConcluso

context Tecnico::getHeatmapImp()
pre: NOT selIntervento.isConcluso

context Tecnico::getInterventiImp()
pre: NOT selIntervento.isConcluso

context Tecnico::modificaImpianto()
pre: NOT selIntervento.isConcluso