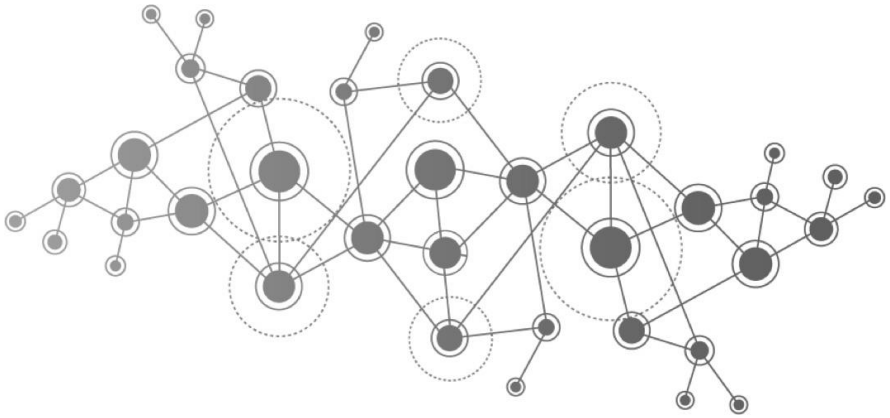
Smart City – Smart Traffic Management

Patrik Brighenti
272841@studenti.unimore.it

Intelligent IoT



UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

A.A 2021/2022

Il progetto

Obiettivo

- Realizzazione di un sistema IoT in grado di favorire la viabilità cittadina

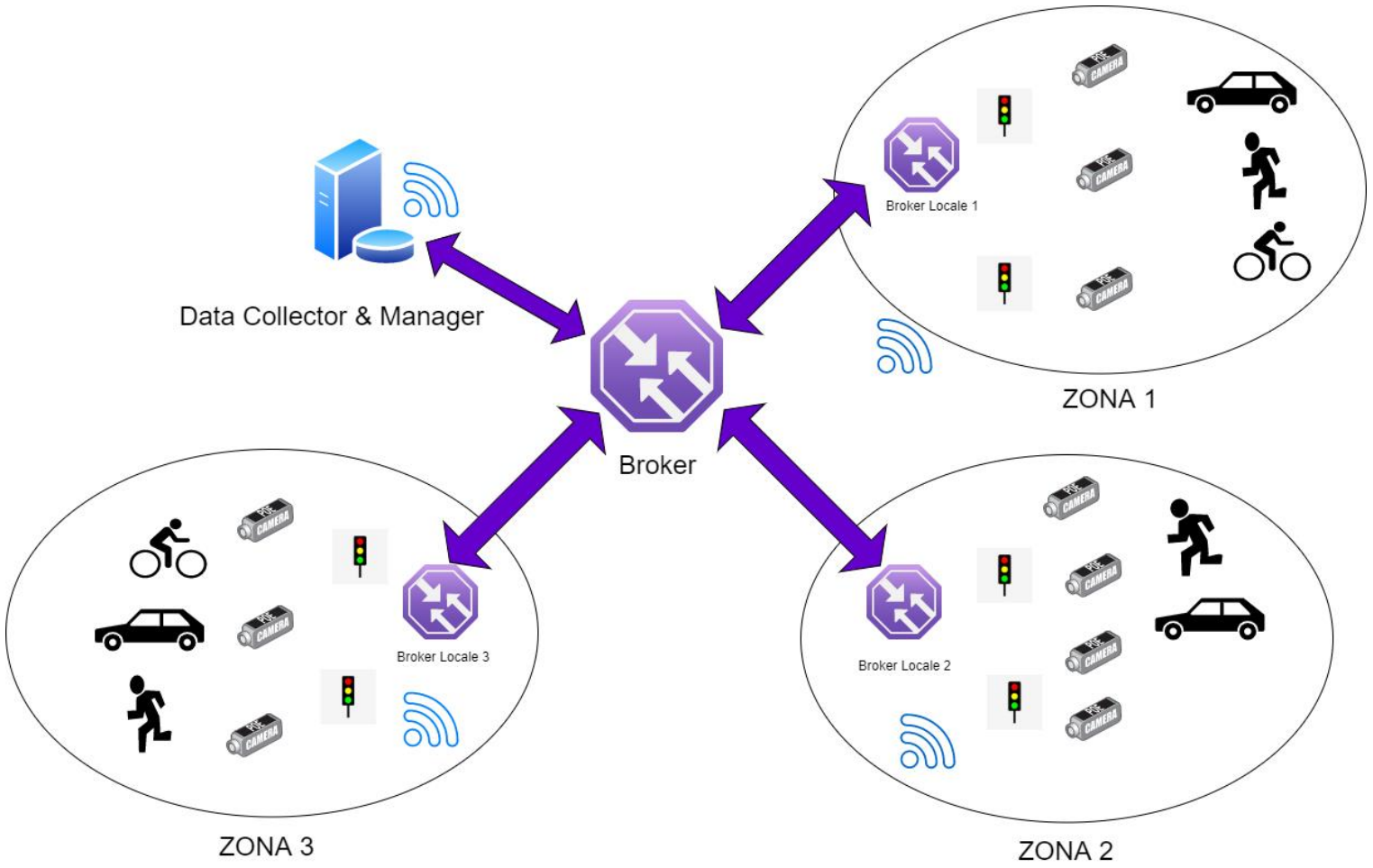
Smart Object del sistema IoT

- Data Collector, Smart Camera & Traffic light

Come

- Se il Manager rileva un traffico eccessivo su un tratto stradale può decidere di ri-bilanciare i tempi dei Traffic Light per favorire la viabilità.

Architettura

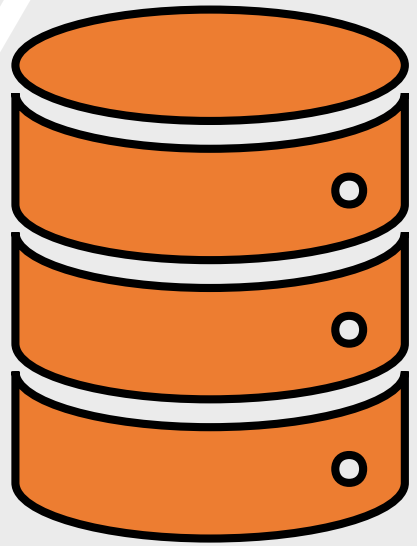


DATA MODELING

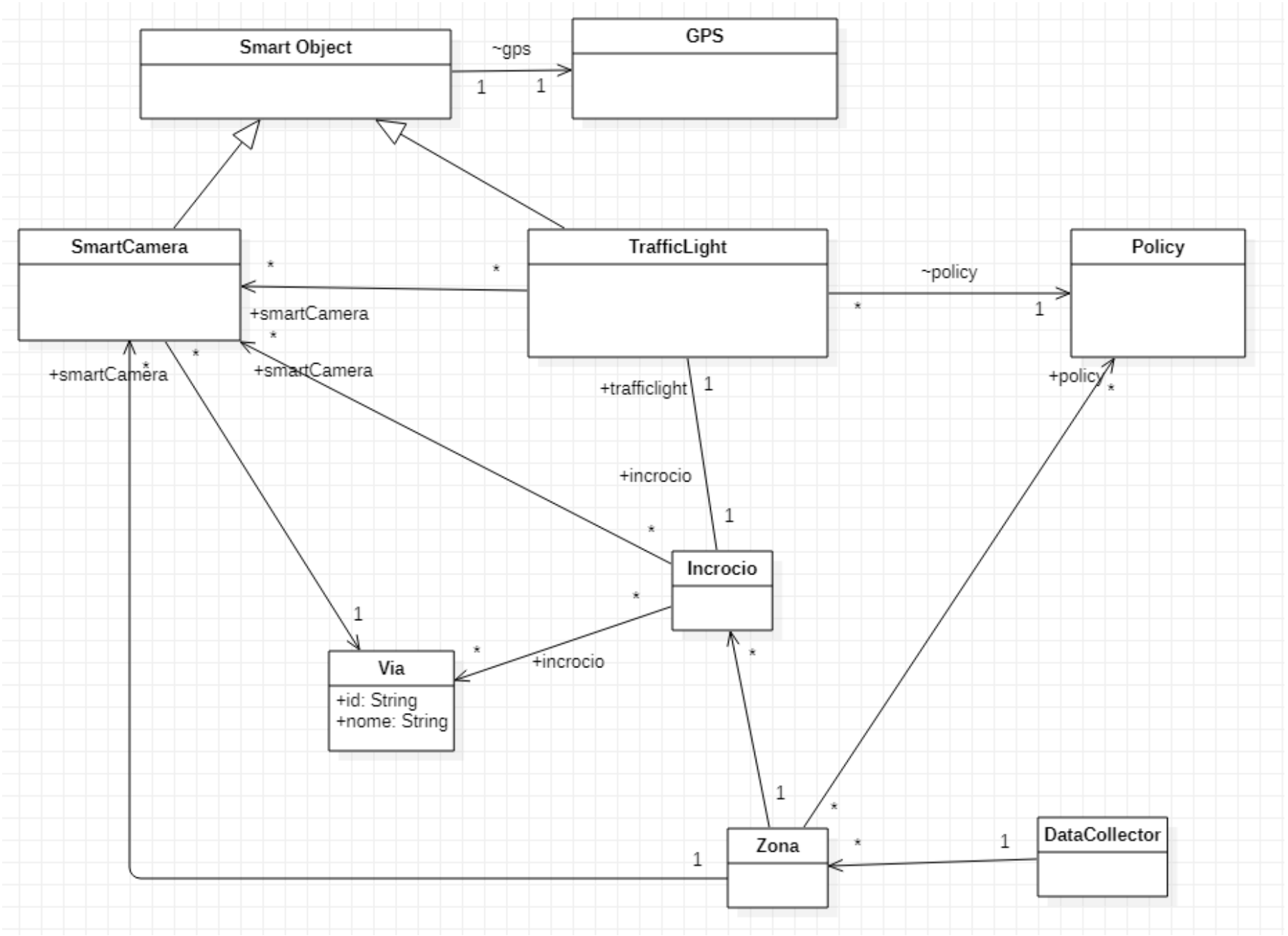
-Schema UML

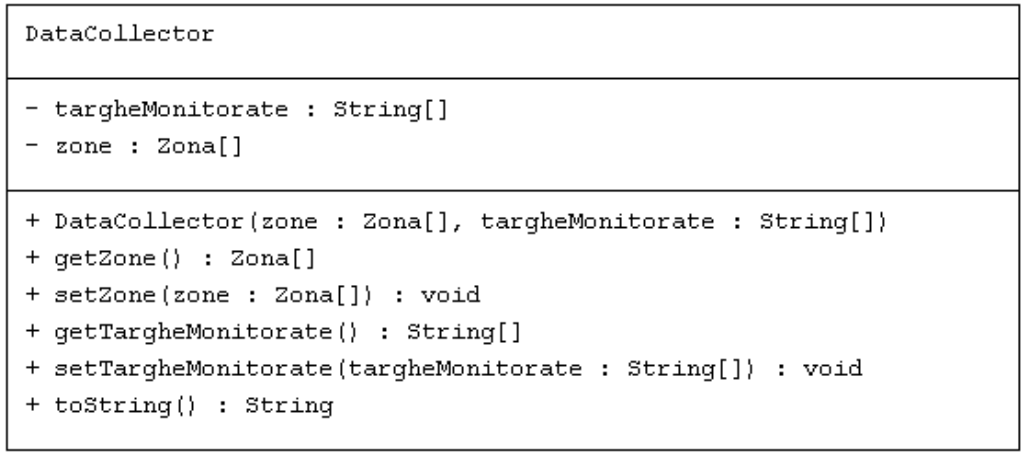
-Struttura Topic

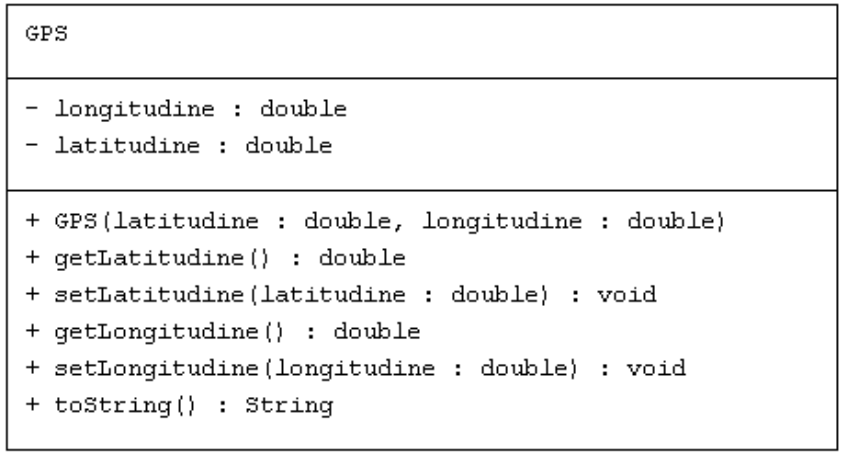
-Struttura Progetto

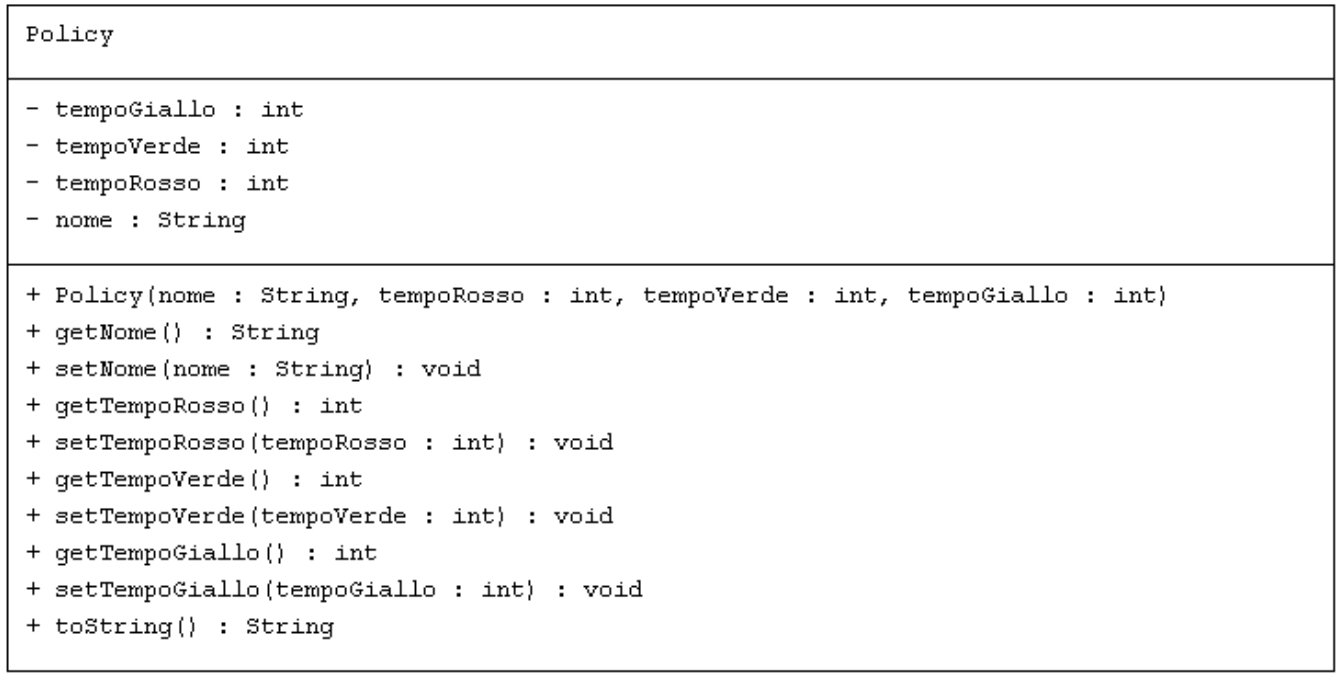


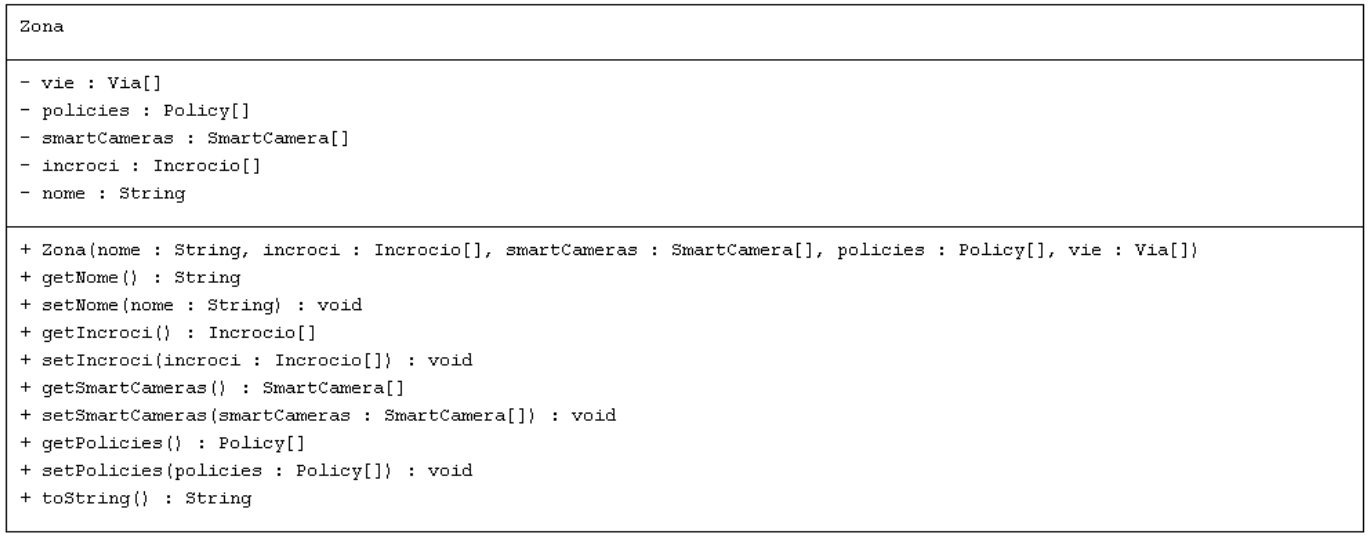
Schema UML

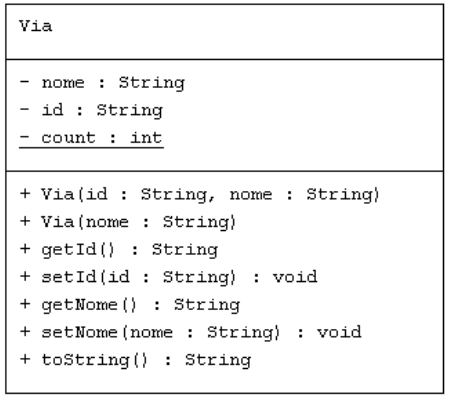

Smart Object
~gps
GPS
1
1
SmartCamera
TrafficLight
Policy
~policy
+smartCamera
+smartCamera
+smartCamera
+trafficlight
+policy
+incrocio
Incrocio
Via
+id: String
+nome: String
+incrocio
Zona
DataCollector

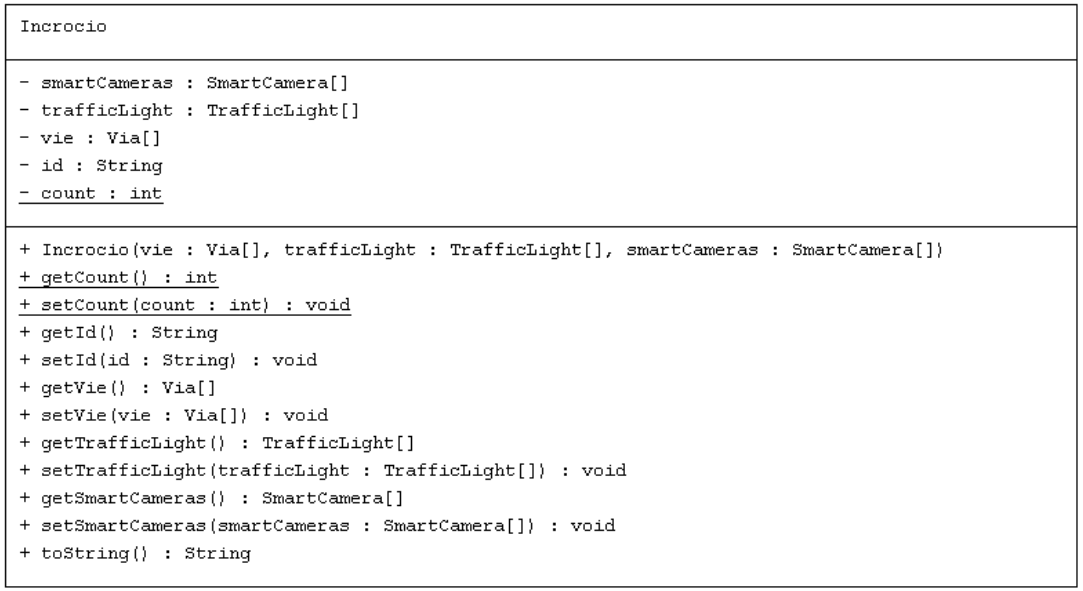

DataCollector
- targheMonitorate : String[]
- zone : Zona[]
+ DataCollector(zone : Zona[], targheMonitorate : String[])
+ getZone() : Zona[]
+ setZone(zone : Zona[]) : void
+ getTargheMonitorate() : String[]
+ setTargheMonitorate(targheMonitorate : String[]) : void
+ toString() : String


GPS
- longitudine : double
- latitudine : double
+ GPS(latitudine : double, longitudine : double)
+ getLatitudine() : double
+ setLatitudine(latitudine : double) : void
+ getLongitudine() : double
+ setLongitudine(longitudine : double) : void
+ toString() : String

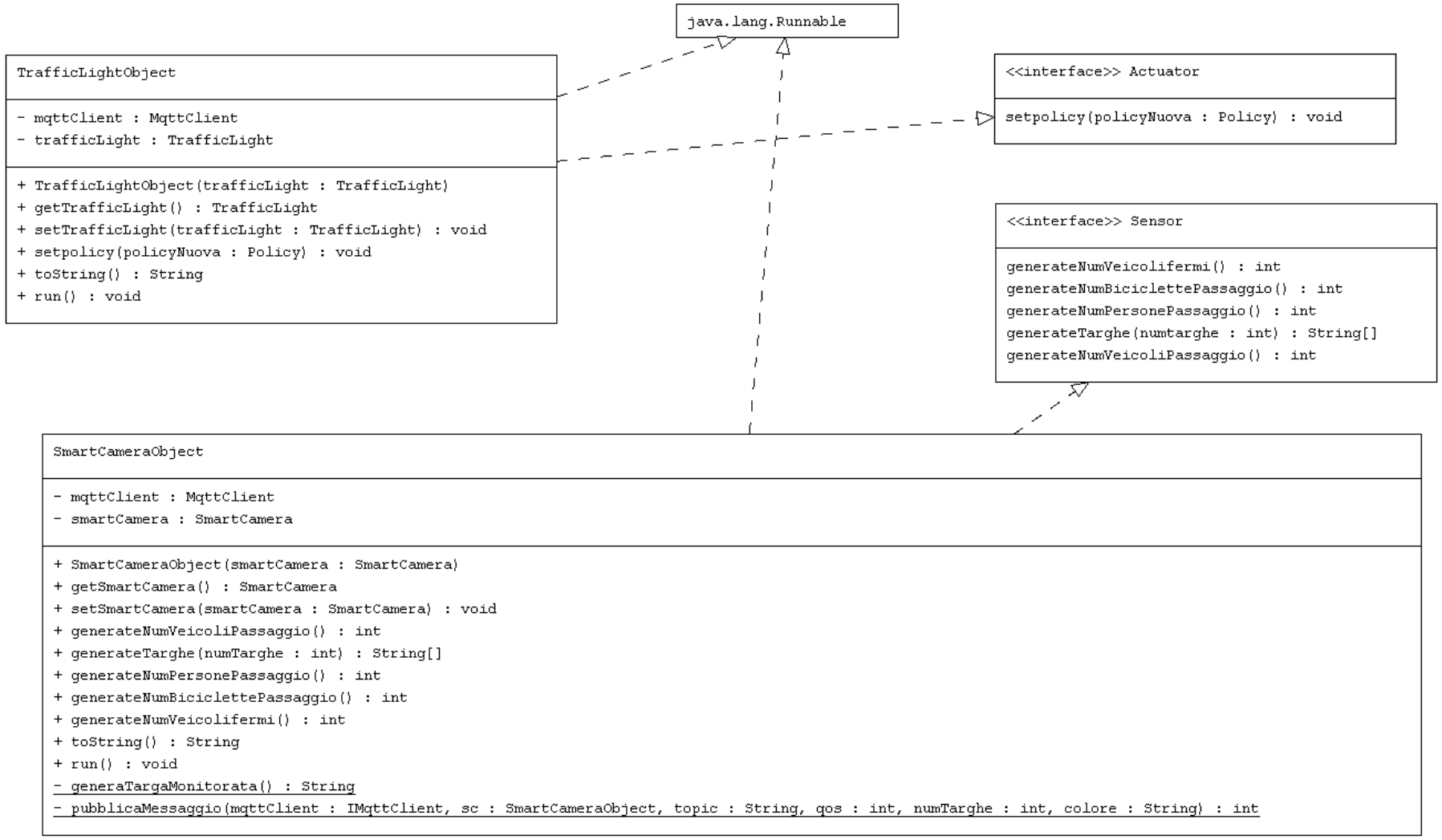

Policy
- tempoGiallo : int
- tempoVerde : int
- tempoRosso : int
- nome : String
+ Policy(nome : String, tempoRosso : int, tempoVerde : int, tempoGiallo : int)
+ getNome() : String
+ setNome(nome : String) : void
+ getTempoRosso() : int
+ setTempoRosso(tempoRosso : int) : void
+ getTempoVerde() : int
+ setTempoVerde(tempoVerde : int) : void
+ getTempoGiallo() : int
+ setTempoGiallo(tempoGiallo : int) : void
+ toString() : String


Zona
- vie : Via[]
- policies : Policy[]
- smartCameras : SmartCamera[]
- incroci : Incrocio[]
- nome : String
+ Zona(nome : String, incroci : Incrocio[], smartCameras : SmartCamera[], policies : Policy[], vie : Via[])
+ getNome() : String
+ setNome(nome : String) : void
+ getIncroci() : Incrocio[]
+ setIncroci(incroci : Incrocio[]) : void
+ getSmartCameras() : SmartCamera[]
+ setSmartCameras(smartCameras : SmartCamera[]) : void
+ getPolicies() : Policy[]
+ setPolicies(policies : Policy[]) : void
+ toString() : String

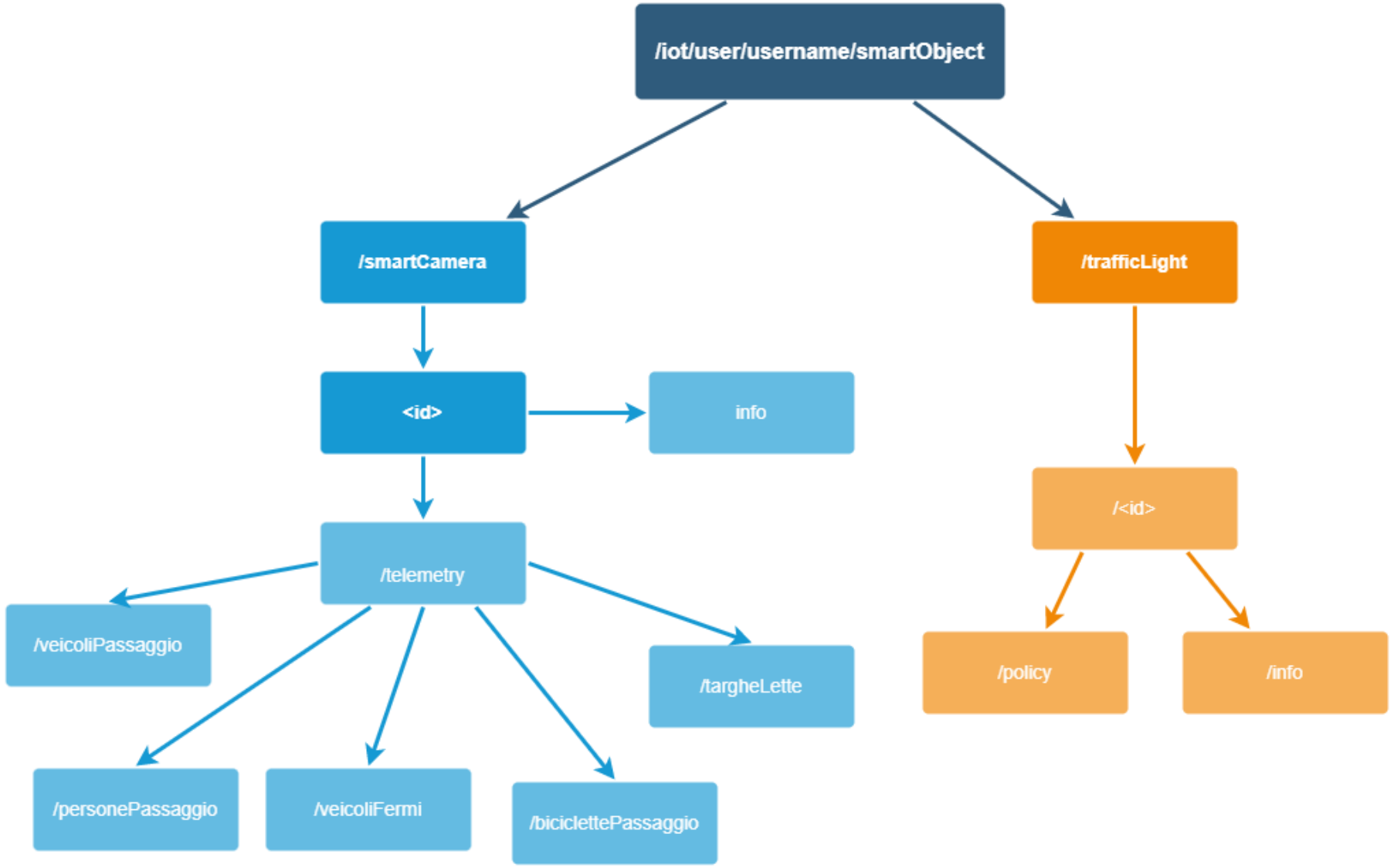

Via
- nome : String
- id : String
- count : int
+ Via(id : String, nome : String)
+ Via(nome : String)
+ getId() : String
+ setId(id : String) : void
+ getNome() : String
+ setNome(nome : String) : void
+ toString() : String


Incrocio
- smartCameras : SmartCamera[]
- trafficLight : TrafficLight[]
- vie : Via[]
- id : String
- count : int
+ Incrocio(vie : Via[], trafficLight : TrafficLight[], smartCameras : SmartCamera[])
+ getCount() : int
+ setCount(count : int) : void
+ getId() : String
+ setId(id : String) : void
+ getVie() : Via[]
+ setVie(vie : Via[]) : void
+ getTrafficLight() : TrafficLight[]
+ setTrafficLight(trafficLight : TrafficLight[]) : void
+ getSmartCameras() : SmartCamera[]
+ setSmartCameras(smartCameras : SmartCamera[]) : void
+ toString() : String

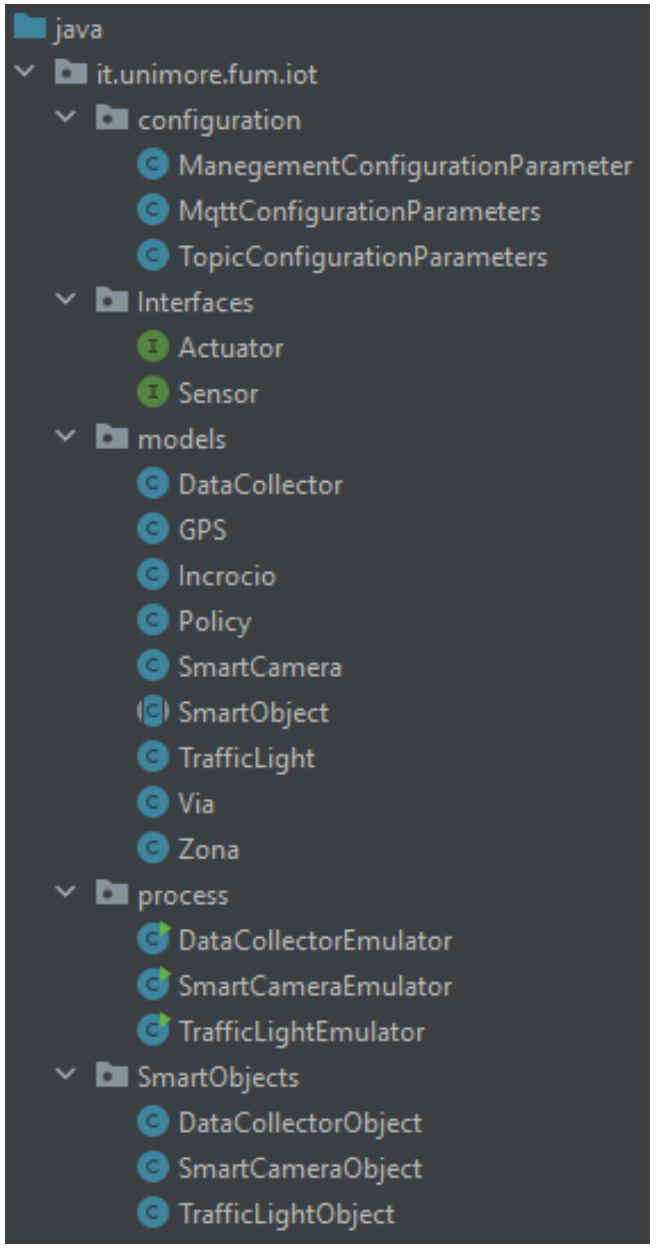

SmartObject
- type : String
- gps : GPS
- id : String
- countTrafficLight : int
- countCamera : int
+ SmartObject(gps : GPS)
+ getCountCamera() : int
+ setCountCamera(countCamera : int) : void
+ incrementCountCamera() : void
+ getCountTrafficLight() : int
+ setCountTrafficLight(countTrafficLight : int) : void
+ incrementCountTrafficLight() : void
+ getId() : String
+ setId(id : String) : void
+ getType() : String
+ setType(type : String) : void
+ getGps() : GPS
+ setGps(gps : GPS) : void
+ toString() : String
TrafficLight
- policyPossibili : Policy[]
- policyAttiva : Policy
+ TrafficLight(gps : GPS, policy : Policy)
+ toString() : String
+ getPolicyAttiva() : Policy
+ setPolicyAttiva(policy : Policy) : void
+ getPolicyPossibili() : Policy[]
+ setPolicyPossibili(policyPossibili : Policy[]) : void
SmartCamera
- via : Via
+ SmartCamera(gps : GPS, via : Via)
+ getVia() : Via
+ setVia(via : Via) : void
+ toString() : String


java.lang.Runnable
TrafficLightObject
- mqttClient : MqttClient
- trafficLight : TrafficLight
+ TrafficLightObject(trafficLight : TrafficLight)
+ getTrafficLight() : TrafficLight
+ setTrafficLight(trafficLight : TrafficLight) : void
+ setpolicy(policyNuova : Policy) : void
+ toString() : String
+ run() : void
<<interface>> Actuator
setpolicy(policyNuova : Policy) : void
<<interface>> Sensor
generateNumVeicolifermi() : int
generateNumBiciclettePassaggio() : int
generateNumPersonePassaggio() : int
generateTarghe(numtarghe : int) : String[]
generateNumVeicoliPassaggio() : int
SmartCameraObject
- mqttClient : MqttClient
- smartCamera : SmartCamera
+ SmartCameraObject(smartCamera : SmartCamera)
+ getSmartCamera() : SmartCamera
+ setSmartCamera(smartCamera : SmartCamera) : void
+ generateNumVeicoliPassaggio() : int
+ generateTarghe(numTarghe : int) : String[]
+ generateNumPersonePassaggio() : int
+ generateNumBiciclettePassaggio() : int
+ generateNumVeicolifermi() : int
+ toString() : String
+ run() : void
- generaTargaMonitorata() : String
- pubblicaMessaggio(mqttClient : IMqttClient, sc : SmartCameraObject, topic : String, qos : int, numTarghe : int, colore : String) : int

Struttura Topic



Struttura Progetto



Classi che definiscono i parametri di configurazione:
-Client MQTT
-Struttura Topic
-Logica Applicativa

Actuator → TrafficLight
Sensor → SmartCamera

Classi che modellano la rappresentazione delle risorse

Classi che permettono di emulare il funzionamento degli oggetti

Classi che modellano gli oggetti smart (oltre alla rappresentazione delle risorse implementano le funzionalità)

Flusso di interazione

Broker

Data Collector

dati sensing

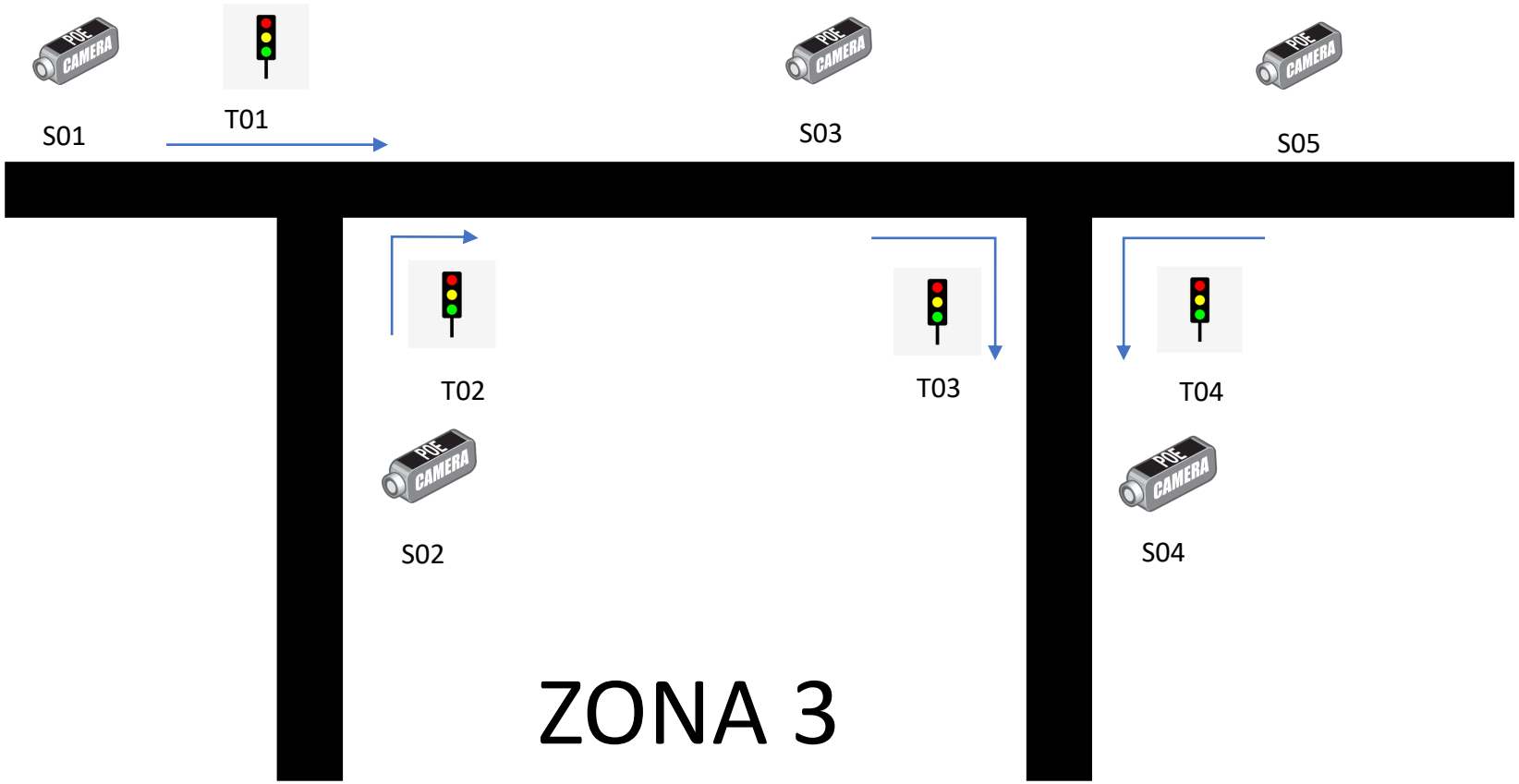
dati sensing

Logica Applicativa

cambio policy

cambio policy

Scenario Applicativo



Considerazioni e sviluppi successivi

- Si è voluto dare una panoramica sulle scelte di progettazione astraendosi dalle varie implementazioni possibili.
- Il progetto è nato dalla modellazione di uno scenario estremamente semplice, ma è stato modellato per poter supportare in futuro l'aggiunta di altri dispositivi e diverse logiche applicative.
- La logica applicativa è solo un esempio, in un contesto di smart city si potrebbe settare una logica che abbia una visione globale su tutte le zone della città (la struttura del progetto permette di farlo).
- Il manager potrebbe comunicare tramite protocollo HTTP con servizi esterni, per esempio per settare la logica applicativa oppure segnalare le targhe monitorate.