Titolo del Progetto:	IPERMOB - Infrastruttura Pervasiva Eterogenea Real-time per il controllo della MOBilità
Tipo di documento:	**Deliverable** ☒ **Documento Interno** ☐ **Altro** _____

Titolo:	Specifiche Funzionali OO4
Data di emissione:	
Redattori:	**Giuseppe Amato, Luigi Costalli, Claudio Gennaro, Mirco Nanni, Salvatore Rinzivillo, Claudio Vairo, Lorenzo Gabrielli, Massimo Zedda.**
Approvatore:	**Claudio Gennaro**
Destinatari:	**ALL**

Il presente documento è stato redatto in coerenza con il Codice Etico e i Principi Generali, in Materia di Proprietà e Protezione dei Risultati, dei Diritti di Accesso e di Riservatezza, elencati nell'Accordo di Collaborazione stipulato dalle Parti

INDICE

PREMESSA

In questo documento viene fornita una descrizione ad alto livello delle funzionalità fornite dall' Obiettivo 4.

Storia Documento

Versione	Data	Motivazioni
1.0 Draft A	22/12/2009	Prima versione del documento proposto per discussione
1.1 Draft B	07/01/2010	Versione del documento comprendente sezione sul S&M DB
1.2 Draft C	07/05/2010	Versione aggiornata API unità mobili, coordinate riferite al grafo
1.3 Draft D	21/10/2010	Versione aggiornata API unità mobili
1.4 Finale	24/2/2011	Versione finale, aggiunta specifiche REST

1 SPECIFICHE FUNZIONALI DELL' OBIETTIVO 4

Scopo generale dell' obiettivo OO4 è fornire l'infrastruttura di data and knowledge management in grado di supportare l'acquisizione continua di dati di sosta e di mobilità dal sistema WSN e l'erogazione di servizi di informazione sulla mobilità sia verso il sistema di monitoraggio real-time che verso i veicoli in movimento.

Tale obiettivo può essere ulteriormente dettagliato come segue:

- fornire una infrastruttura di data management per la rappresentazione dello stato corrente dell'ambiente monitorato. Tale infrastruttura recepisce in modo continuo (streaming) dati relativi all'ambiente monitorato dal sistema WSN e corrispondentemente aggiorna l'insieme di strutture dati (indicatori sullo stato delle risorse) che servono in modo continuo le query dei servizi verso l'utente finale.
- fornire una infrastruttura di storicizzazione dei dati osservati per supportare il monitoraggio dell'evoluzione del sistema al fine di
  - alimentare il sistema di supporto alle decisioni del pianificatore del traffico e del sistema dei parcheggi;
  - supportare la validazione e l'aggiornamento del sistema di indicatori necessari al sistema di monitoraggio ed informazione real-time;
- costruire un modello di mobilità del sistema monitorato in grado di ottimizzare l'allocazione delle risorse di parcheggio e di viabilità in funzione ad una stima dei flussi di traffico in ingresso, uscita ed interni ottenuta in tempo reale o quasi reale. Tale modello di mobilità identifica le soglie di rischio degli indicatori sullo stato delle risorse che verranno utilizzate dai servizi di instradamento verso l'utente finale;
- capire l'interazione del sistema monitorato con un modello di mobilità dell'ambiente circostante a quello monitorato mediante l'uso di una nuova generazione di matrici O-D adattive rispetto ai dati di sosta, flusso e percorso desunti dal sistema monitorato e da altre fonti esterne (tracce GPS e tracce originate dalla rete GSM/GPS).

Il risultato di questo obiettivo operativo è pertanto duplice. 1) Verrà creata un'infrastruttura per l'analisi dei dati e la risposta in real-time alle richieste. 2) Verrà sviluppata una metodologia per l'analisi dei dati di traffico. In Figura 1 sono mostrati i componenti dell'OO4, all'interno dell'architettura del progetto.

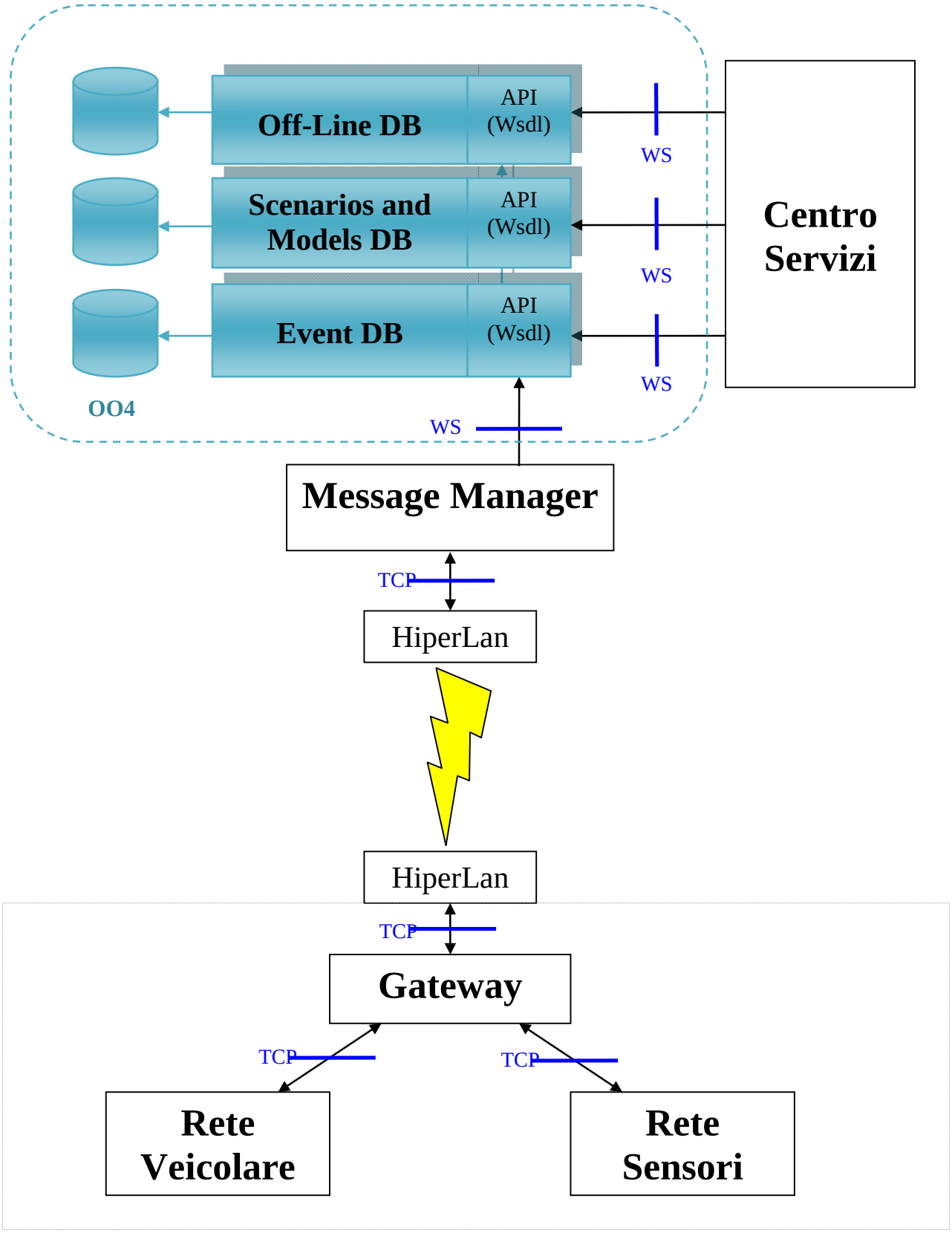


Figura 1 Architettura OO4

1.1 Event - DataBase

L'Event Database (o On-Line Database) ha l'obiettivo di garantire in tempo reale la persistenza degli eventi occorsi nei vari sensori della rete di monitoraggio. In particolare, Event Database:

- riceve in modo real-time dati di sosta dalle varie risorse di parcheggio dell'ambiente monitorato e flussi veicolari delle risorse stradali nell'ambiente monitorato;
- Aggiorna le strutture dati funzionali al sistema di monitoraggio real-time ed al sistema di infomobilità;
- Aggiorna l'informazione necessaria al sistema on-line e prepara le informazioni da inviare al sistema Off-Line Database;
- Risponde alle interrogazioni riguardanti i dati real-time ricevuti, per soddisfare le richieste in arrivo dal centro di controllo (OO5).

1.1.1 Inserimento eventi

1.1.1.1 Vehicle transit detection

- **vehiclesDetected**(time, period, **listof**<sensorId, numFlows, **listof**<flowId, numVehicles, classA, classB, classC, classD, speed>>)
  - *Inserisce un evento riguardo la rilevazione di una serie di veicoli transitati da parte di più sensori (posizioni del sensori conosciute dall'on-line DB) . Ogni sensore controlla più flussi (un flusso è una corsia) e ogni flusso identifica 4 tipi di classi di veicoli identificati con classe A, classe B, classe C* e *classe D.*
  - ***time***: time-stamp dell'evento.
  - ***period***: periodo di osservazione a cui si riferisce la rilevazione in msec.
  - ***sensorId***: identificatore unico del sensore.
  - ***numFlows***: numero di flussi osservati dal sensore.
  - ***flowId***: id del flusso
  - **numVehicles**: Numero totale veicoli transitati nel ***period.***
  - ***classA***: Percentuale veicoli di classe A nel tempo di osservazione.
  - ***classB***: Percentuale veicoli di classe B nel tempo di osservazione.
  - ***classC***: Percentuale veicoli di classe C nel tempo di osservazione.
  - ***classD***: Percentuale veicoli di classe D nel tempo di osservazione.
  - ***speed***: Velocità media veicoli.

1.1.1.2 Vehicle position update

- ***vehiclePositionUpdate***(sessionId, vehicleId, longitude, latitude, arcId, segmentOrder, time)
    - *Inserisce un evento riguardo l'aggiornamento della posizione di un veicolo da parte di sensore mobile installato a bordo del veicolo. Verifica che la sessione sia ancora valida, in caso contrario l'invocazione restituisce un messaggio di errore. I veicoli dovranno fornire le coordinate corrette del veicolo posto sul grafo della rete veicolare (come* longitude/latitude*), insieme all'id dell'arco del grafo dove il veicolo è posto, e il numero del segmento all'interno dell'arco. Il grafo è fornito dalle API dello Scenarios and Models DB (Paragrafo 1.3).*
    - ***sessionId***: identificatore unico della sessione associato al veicolo
    - ***vehicleId***: identificatore unico del veicolo (es., numero di targa).
    - ***longitude, latitude***: posizione del veicolo, coordinate geografiche corrette.
    - ***arcId***: posizione del veicolo, id arco del grafo.
    - ***segmentOrder***: numero del segmento all'interno dell'arco
    - ***time***: time-stamp dell'evento.

1.1.1.3 Vehicle travel time detection

- **vehicleTravelTimeDetected**(sessionId, vehicleId, ttid, travelTime, time)
    - *Inserisce un evento riguardo il tempo di transito di un veicolo da parte di un sensore mobile installato a bordo del veicolo rispetto ad un tratto di strada prefissato sul grafo della rete veicolare identificato univocamente con l'id ttid. Verifica che la sessione si ancora valida, in caso contrario l'invocazione restituisce un messaggio di errore.*
    - ***sessionId***: identificatore unico della sessione associato al veicolo
    - ***vehicleId***: identificatore unico del veicolo (es., numero di targa).
    - ***ttid***: id unico del tratto di strada del travel time.
    - **travelTime**: tempo di transito in msec.
    - ***time***: time-stamp dell'evento.

1.1.1.4 Parking slot monitoring

- **updateParkingSlot**(time, **listof<**sensorId, numSlots, **listof**< slotId, state>>)
    - *Aggiorna lo stato di un certo numero di stalli monitorati da un sensore di un parcheggio (posizione del sensore conosciuta dall'on-line DB). Ogni invocazione aggiorna lo stato di tutti gli stalli monitorati da più sensori. Gli stati vengono aggiornati periodicamente per cui questo metodo verifica lo stato attuale modifica lo stato di uno stato di uno stallo solo se effettivamente cambiato. Il parametro status non rappresenta lo stato reale dello stallo ma una stima che viene valutata dal singolo sensore.*

Infatti, ogni stallo può essere monitorato da più sensori (multiview). Attraverso la specifica degli errori sistematici ed di una funzione che viene fornita al sistema l'updateParkingSlot decide lo stato finale dello stallo. Questo metodo invia un Trigger al centro servizi nel caso di cambio di stato di uno o piàù stalli.

- ***time***: time-stamp dell'evento.
- ***sensorId***: identificatore unico del sensore di un parcheggio.
- ***numSlots***: numero di stalli monitorati.
- ***slotId***: identificatore dello slot
- ***state***: stato degli stallo monitorato, ha un valore codificato con un intero tra 0 e 255.

1.1.2 Interrogazioni

1.1.2.1 Car Park status

- **getParkStatus**(parkId)
  - *restituisce lo stato di tutti gli stalli di una dato parcheggio*
  - ***parkId***: identificatore unico del parcheggio.
- **getParkedVehicles**(sessionId, veichleId, parkId)
  - *restituisce il numero totale di veicoli parcheggiati in un dato parcheggio. Verifica che la sessione si ancora valida, in caso contrario l'invocazione restituisce un messaggio di errore (vedi sotto per la gestione delle sessioni).*
  - ***sessionId***: identificatore della sessione.
  - ***vehicleId***: identificatore unico del veicolo.
  - ***parkId***: identificatore unico del parcheggio.

1.1.2.2 Vehicle Flow

- ***getVehicleFlow***(archId, startTime, endTime)
  - *restituisce il numero di veicoli transitati in un dato arco del grafo all'interno di una certa finestra temporale. L'arco di un grafo rappresenta un tratto di strada e può contenere uno o più flussi vale a dire corsie. Questo metodo restituisce il numero di veicoli, la loro velocità, totale e la percentuale per ogni classe.*
  - ***archId***: identificatore unico dell'arco
  - ***startTime***: tempo di inizio della finestra temporale
  - ***endTime***: tempo di fine della finestra temporale
- ***getEstimatedTravelTime****(sessionId, veichleId,* ttid*)*

- *restituisce il tempo medio di transito dei veicoli tra due posizioni da parte di un sensore mobile installato a bordo del veicolo rispetto ad un tratto di strada prefissato sul grafo della rete veicolare identificato univocamente con l'id ttid. Verifica che la sessione si ancora valida, in caso contrario l'invocazione restituisce un messaggio di errore.*
- ***sessionId***: identificatore unico della sessione associato al veicolo
- ***vehicleId***: identificatore unico del veicolo (es., numero di targa).
- ***ttid***: id unico del tratto di strada del travel time.

1.1.2.3 Triggers

- ***eventParkingSlotStatusChange(listof**<slotId,state>)*
  - *metodo implementato dal client, chiamato dall'on-line DB in caso di attivazione del trigger "cambio di stato di una lista di stalli"*
  - ***slotId***: identificatore unico dello stallo.
  - ***state***: stato del posto di parcheggio (pieno o vuoto).
- ***eventFlowStatusUpdate***(period, **listof**<arcId, numVehicles, classA, classB, classC, classD, speed>)
  - *metodo implementato dal client, chiamato dall'on-line DB che notifica il numero di veicoli transitati in un dato arco.*
  - ***period***: periodo di osservazione a cui si riferisce la rilevazione in msec.
  - ***sensorId***: identificatore unico del sensore.
  - ***numFlows***: numero di flussi osservati dal sensore.
  - ***flowId***: id del flusso
  - **numVehicles**: Numero totale veicoli transitati nel ***period***.
  - ***classA***: Percentuale veicoli di classe A nel tempo di osservazione.
  - ***classB***: Percentuale veicoli di classe B nel tempo di osservazione.
  - ***classC***: Percentuale veicoli di classe C nel tempo di osservazione.
  - ***classD***: Percentuale veicoli di classe D nel tempo di osservazione.
  - ***speed***: Velocità media veicoli.
- ***eventPositionUpdate****(vehicleId, longitude, latitude, arcId, segmentOrder, time)*
  - *metodo implementato dal client, chiamato dall'on-line DB in caso di attivazione del trigger "rilevazione di posizione veicolo dal sensore mobile".*
  - ***vehicleId***: identificatore unico del veicolo (es., numero di targa).
  - ***longitude, latitude***: posizione del veicolo, coordinate geografiche corrette.
  - ***arcId***: posizione del veicolo, id arco del grafo.
  - ***segmentOrder***: numero del segmento all'interno dell'arco
  - ***time***: time-stamp dell'evento.

- ***eventAlarm***(time, sensorId, eventType)
  - *metodo implementato dal client, chiamato dall'on-line DB in caso di attivazione del trigger "segnalazione allarme".*
  - ***eventType****: tipo di allarme; il valore 1 indica un errore generico.*
  - ***sensorId***: identificatore unico del sensore.
  - ***time***: time-stamp dell'evento.

1.1.3 Accounting

- **createAccount**(userId, password, profileId, status, details)
  - *crea un nuovo utente autorizzato ad avere accesso all'on-line DB*
  - ***userId/passwd***: credenziali dell'utente.
  - ***profileId***: id del profile utente.
  - ***status***: idenfitica lo stato della richiesta che può essere: *pending, created, refused* o *deleted.*
  - ***details***: dettagli dell'utente
- **getAccount**(userId, status)
  - *restituisce tutte le informazioni riguardanti un utente ad eccezione di passwd*
  - ***userId***: id utente
  - ***status***: idenfitica lo stato della richiesta che può essere: *pending, created, refused* o *deleted.*
- **getAccount**(status)
  - *restituisce tutti gli userid degli utenti che hanno uno specifico status.*
  - ***userId***: id utente
  - ***status***: idenfitica lo stato della richiesta che può essere: *pending, created, refused* o *deleted.*
- **updateAccount**(userId, password, profileId, status, details)
  - *aggiorna i dati di un account*
  - ***userId/passwd***: credenziali dell'utente.
  - ***profileId***: id del profile utente.
  - ***status***: idenfitica lo stato della richiesta che può essere: *pending, created, refused* o *deleted.*
  - ***details***: dettagli dell'utente
- **checkAccount**(userId, password, status)
  - *verifica l'esistenza di un utente con le credenziali date*
  - ***userId/passwd***: credenziali dell'utente.

- ***status***: idenfitica lo stato della richiesta che può essere: *pending, created, refused* o *deleted.*

- **sessionRequest**(userId, password, vehicleId)
  - *restituisce una nuova sessione (sessionId) associata ad un utente del DB. Prima verifica se la tripla vehicleId/user/pass è valida. La sessione dura di default 24 ore*. Il centro di servizi ha un sessionId che non scade mai.
  - ***userId/passwd***: credenziali dell'utente.
  - ***vehicleId***: identificatore unico del veicolo.
- **logout**(sessionId, vehicleId)
  - *cancella una sessione di un utente.*
  - ***sessionId***: id della sessione.
  - ***vehicleId***: identificatore unico del veicolo.

1.1.4 Gestione allarmi

- **alarm**(sensorId, eventType, time)
  - *Inserisce un evento riguardo la rilevazione di un errore a livello di rete di sensori da parte di un sensore (posizione del sensore conosciuta dall'on-line DB). L'invocazione produce sempre una chiamata ad uno specifico servizio del centro di servizi, che notifica l'allarme.*
  - ***eventType****: tipo di allarme; il valore 1 indica un errore generico.*
  - ***sensorId***: identificatore unico del sensore.
  - ***time***: time-stamp dell'evento.

1.2 Off Line DataBase

L'Off Line Database (o Data Warehouse) è popolato con lo storico dei dati raccolti dall'Event Database. Periodicamente (ad esempio ogni giorno) il Data Warehouse colleziona i dati raccolti dall'On-line Database. I dati elementari, detti anche *fatti*, sono aggregati e memorizzati secondo diverse misure e dimensioni che vengono definite al momento della definizione dello schema del Data Warehouse. Questa aggregazione permette un accesso efficiente alle statistiche di alto livello dei dati e una gestione più compatta dell'informazione memorizzata, permettendo la memorizzazione di lunghi periodi di raccolta (es. mesi, anni). Il singolo fatto memorizzato sul Data Warehouse può essere aggregato su diverse *dimensioni*. Una dimensione rappresenta una vista sui dati utilizzando criteri differenti. Riprendendo l'esempio del conteggio degli stalli occupati, è possibile aggregare il conteggio su diverse dimensioni, come ad esempio la locazione della piazzola o la finestra temporale. Per le diverse aggregazioni definite sui dati, è possibile definire diversi livelli di aggregazione. Ad esempio, il conteggio delle soste sugli stalli può essere effettuato su diversi periodi temporali, aggregando per settimana, mese, semestre, anno, ecc. Infine, per ogni aggregato è possibile definire delle *misure*, ovvero funzioni che

mappano il risultato dell'aggregazione su un valore numerico (es. somma, media, varianza, ecc.).

1.2.1 Querying

- getCubeList()
  - Ritorna l'elenco dei cubi del DW sui quali è possibile effettuare le interrogazioni.
- getDimensionsList(cube)
  - Ritorna la lista delle dimensioni definite su un particolare cubo del DW.
- getMeasuresList(cube)
  - Ritorna una lista di possibili misure definite sul cubo specificato.
- getLevelsForDimension(dimension)
  - Ritorna una lista di livelli disponibili per la dimensione specificata
- getPivotTable(dimension1, [livello1], dimension2, [livello2], [measure], constraint)
  - Ritorna una matrice bidimensionale contenente le aggregazioni delle due dimensioni specificate secondo la misura scelta. Le righe e le colonne della matrice corrispondono alle due dimensioni. Il livello scelto per ogni dimensione determina il livello di aggregazione usato e, come conseguenza, il numero di righe e colonne della matrice. Eventuali vincoli sui dati da considerare possono essere specificati tramite il parametro "constraint".
- getParkingHistory(park)
  - Ritorna il numero di autoveicoli che hanno sostato in un parcheggio specificato.
- getArcHistory(arc)
  - Ritorna il numero di veicoli che sono transitati su un arco in uno specifico intervallo di tempo; ritorna inoltre la velocità media e un valore numerico percentuale relativo ad ogni *Classe* di veicoli. La *Classe* di appartenenza del veicolo viene assegnata dal sistema di rilevazione automatico dei flussi veicolari.

Per chiarire meglio il processo di interrogazione dell'Off-line DB forniamo di seguito alcuni esempi di interrogazione.

Esempio 1. Determinare la presenze di autoveicoli per mese in un parcheggio durante l'ultimo anno. Le dimensioni da considerare sono "parcheggi" (livello di granularità minimo) e "tempo" (livello di granularità "mese"). La risposta dell'Off-line DB all'invocazione getPivotTable("parcheggi", "lvl0","tempo","mese", "sum(presence)", "tempo within 2009") sarà una tabella contenente, per ogni cella, la somma dei veicoli che hanno sostato nel parcheggio corrispondente alla riga della matrice nell'intervallo di tempo corrispondente alla colonna.

Esempio 2. Determinare la velocità media dei veicoli, per ogni ora, passanti per i sensori di uno specifico quartiere nell'ultima settimana. Le dimensioni considerate sono "sensore" (livello di granularità minimo) e "tempo" (livello di granularità "ora"). La tabella risultante dall'invocazione del metodo getPivotTable("sensore", "lvl0", "tempo", "ora", "avg(velocity)","week = 49 AND year=2009) avrà una riga corrispondente ad ogni sensore e una colonna corrispondente ad ogni ora della settimana. Il valore di ogni cella conterrà la velocità media corrispondente ai valori delle due dimensioni delle due righe.

1.3 Scenarios and Models DB

I modelli di simulazione dei trasporti permettono di stimare I flussi sugli archi di una rete come pure i cammini utilizzati dagli utenti sulla base dei conteggi di traffico registrati in un sottoinsieme di sezioni. Essi permettono inoltre di aggiornare una matrice O-D nota a priori. A tal fine la base di dati in esame sarà in grado di fornire le informazioni relative all'assetto della domanda e dell'offerta di trasporto nell'area di studio note a priori ed immagazzinare i risultati delle simulazione al fine di rendere disponibile un archivio storico. E' possibile suddividere i dati che popoleranno il S&M DB in due macro famiglie.

- **Scenario di Base**
  1. grafo rappresentante l'offerta di trasporto stradale in termini di infrastruttura;
  2. zonizzazione del territorio;
  3. matrici O-D iniziale.

- **Risultati delle simulazioni ed aggiornamenti**
  1. storico dei flussi sugli archi dello scenario di riferimento;
  2. storico dei tempi di percorrenza degli archi dello scenario di riferimento;
  3. storico delle matrici o-d aggiornate;

1.3.1 Querying

Tali dati saranno indistintamente consultabili tramite le seguenti interrogazioni:

- *getGraph ()*
  - restituisce la lista degli archi del grafo formato dai seguenti campi:
    - objectid: identificativo dell'arco
    - a: nodo di partenza
    - b: nodo di arrivo
    - cod_scen: codice scenario

- *GetZones ()*
  - restituisce la zonizzazione del territorio formato dai seguenti campi:
    - id: identificativo della zona
    - node: identificativo del nodo
    - addetti: totale addetti della zona
    - popolazione: totale popolazione della zona
    - geometry: geometria della zona

- *getODMatrix ()*
  - restituisce la matrice origine destinazione formata dai seguenti campi:
    - id: indice
    - o: zona di origine
    - d: zona di destinazione
    - flusso: spostamenti da zona a zona
    - codice_mat: codice di scenario

- *getODMatrixCatalog ()*
  - restituisce il catalogo versioni delle matrici o-d:
    - codice_mat: identificativo delle matrici

- *getGraphCatalog ()*
  - restituisce il catalogo delle versioni del grafo:
    - cod_scen: identificativo del grafo

- *getArcFlow (objectid)*
  - restituisce il flusso dell'arco selezionato:
    - objectid: identificativo dell'arco
    - a: nodo di partenza
    - b: nodo di arrivo
    - cod_scen: codice dello scenario
    - name: nome dell'arco
    - flow: valore orario del flusso simulato

- *getArcTravelTime (objectid)*
  - restituisce il tempo di percorrenza dell'arco selezionato*;*

2 APPENDICE - DEFINIZIONE DELLE INTERFACCE REST

2.1 updateParkingSlot(time, List<sensorId, numSlots, List<slotId, state>>)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/parkingStatus

Request format:

```
@XmlRootElement(name = "updateParking")
public class UpdateParking {

   @XmlElement(name = "time")
   public long time;

   @XmlElement(name = "sensor")
   public List<ParkingSensorInfo> parkingSensorList;
}

@XmlRootElement(name = "parkingSensor")
public class ParkingSensorInfo {

   @XmlElement(name = "sid")
   public String sensorId;

   @XmlElement(name = "slot")
   public List<ParkingSlotInfo> parkingSlotList;
}

@XmlRootElement(name = "parkingSlot")
public class ParkingSlotInfo {

   @XmlElement(name = "slid")
   public int slotId;
```

```
    @XmlElement(name = "state")
    public int state;
}
```

Response Format:

SUCCESS: “HTTP 200 OK” with an optional String in the body.

FAIL: “HTTP 500 Internal Server Error” with the String containing the error in the body.

2.2 getParkStatus(parkId)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/parkingStatus

?parkId=value

Request format:

int parkId

Response Format:

```
@XmlRootElement(name = "parkStatusResponse")
public class ParkStatusResponse {

    @XmlElement(name = "slot")
    public List<ParkingSlotInfo> parkingSlotList;
}
```

2.3 getParkedVehicles(sessionId, vechicleId, parkId)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/parkingStatus/parkId

?sessionId=value&vehicleId=value&parkId=value

Request format:

int sessionId, int vehicleId, int parkId

Response Format:

```
@XmlRootElement(name = "parkedVehiclesResponse")
public class ParkedVehiclesResponse {

   @XmlElement(name = "totalSlots")
   public int totalSlots;

   @XmlElement(name = "busySlots")
   public int busySlots;
}
```

2.4 vehiclesDetected(time, period, List<sensorId, numFlows, List<flowId, numVehicles, classA, classB, classC, classD, speed>>)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/vehiclesDetection

Request format:

```
@XmlRootElement(name = "vehiclesDetection")
public class VehiclesDetection {

   @XmlElement(name = "time")
   public long time;

   @XmlElement(name = "period")
   public int period;

   @XmlElement(name = "sensor")
   public List<VehicleFlowSensorInfo> vehicleFlowSensorList;
}

@XmlRootElement(name = "vehicleFlowSensor")
```

```
public class VehicleFlowSensorInfo {

    @XmlElement(name = "sid")
    public String sensorId;

    @XmlElement(name = "flow")
    public List<VehicleFlowInfo> vehicleFlowList;
}

@XmlRootElement(name = "vehicleFlow")
public class VehicleFlowInfo {

    @XmlElement(name = "fid")
    public int flowId;

    @XmlElement(name = "numVehicles")
    public int numVehicles;

    @XmlElement(name = "classA")
    public int classA;

    @XmlElement(name = "classB")
    public int classB;

    @XmlElement(name = "classC")
    public int classC;

    @XmlElement(name = "classD")
    public int classD;

    @XmlElement(name = "speed")
    public int speed;
}
```

Response Format:

SUCCESS: “HTTP 200 OK” with an optional String in the body.

FAIL: “HTTP 500 Internal Server Error” with the String containing the error in the body.

2.5 getVehicleFlow(arcId, startTime, endTime)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/vehiclesDetection

?arcId=value&startTime=value&endTime=value

Request format:

int arcId, long startTime, long endTime

Response Format:

```
@XmlRootElement(name = "vehicleFlowResponse")
public class VehicleFlowResponse {

   @XmlElement(name = "numVehicles")
   public int numVehicles;

   @XmlElement(name = "classA")
   public int classA;

   @XmlElement(name = "classB")
   public int classB;

   @XmlElement(name = "classC")
   public int classC;

   @XmlElement(name = "classD")
   public int classD;

   @XmlElement(name = "speed")
   public int speed;
```

```
}
```

2.6 vehiclePositionUpdate(sessionId, vehicleId, longitude, latitude, arcId, segmentOrder, time)

POST
http://ipermob.isti.cnr.it:8080/EventDB/resources/vehiclesDetection/positionUpdate

Request format:

```
@XmlRootElement(name = "positionUpdate")
public class PositionUpdate {

    @XmlElement(name = "seid")
    public int sessionId;

    @XmlElement(name = "vid")
    public int vehicleId;

    @XmlElement(name = "longitude")
    public float longitude;

    @XmlElement(name = "latitude")
    public float latitude;

    @XmlElement(name = "aid")
    public int arcId;

    @XmlElement(name = "segord")
    public int segmentOrder;

    @XmlElement(name = "time")
    public long time;
}
```

Response Format:

SUCCESS: “HTTP 200 OK” with an optional String in the body.

FAIL: “HTTP 500 Internal Server Error” with the String containing the error in the body.

2.7 vehicleTravelTimeDetected(sessionId, vehicleId, ttid, travelTime, time)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/vehiclesTravelTime

Request format:

```
@XmlRootElement(name = "travelTimeUpdate")
public class TravelTimeUpdate {

    @XmlElement(name = "seid")
    public int sessionId;

    @XmlElement(name = "vid")
    public int vehicleId;

    @XmlElement(name = "ttid")
    public int ttid;

    @XmlElement(name = "travelTime")
    public int travelTime;

    @XmlElement(name = "time")
    public long time;
}
```

Response Format:

SUCCESS: “HTTP 200 OK” with an optional String in the body.

FAIL: “HTTP 500 Internal Server Error” with the String containing the error in the body.

2.8 getEstimatedTravelTime(sessionId, vehicleId, ttid)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/vehiclesTravelTime
?sessionId=value&vehicleId=value&ttId=value

Request format:

int sessionId, int vehicleId, int parkId

Response Format:

```
@XmlRootElement(name = "estimatedTravelTimeResponse")
public class EstimatedTravelTimeResponse {

    @XmlElement(name = "vid")
    public int vehicleId;

    @XmlElement(name = "ttid")
    public int ttid;

@XmlElement(name = "estimatedTravelTime")
    public int estimatedTravelTime;
}
```

2.9 createAccount(userId, password, profileId, status, details)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/create

Request format:

```
@XmlType(name = "user")
@XmlRootElement(name = "user")
public class User {
```

```
    public final static Integer PENDING_STATUS   = 0;
    public final static Integer CREATED_STATUS  = 1;
    public final static Integer REFUSED_STATUS  = 2;
    public final static Integer DELETED_STATUS   = 3;

    @XmlElement(name = "userId", nillable = false, required = true)
    public String userId;

    @XmlElement(name = "password", required = false)
    public String password;

    @XmlElement(name = "profileId", required = false)
    public Integer profileId;

    @XmlElement(name = "status", required = false)
    public Integer status;

    @XmlElement(name = "details", required = false)
    public String details;
}
```

Response Format:

```
@XmlType(name = "response")
@XmlRootElement(name = "response")
public class GenericResponse {

    public static final int NO_ERROR = 0;
    public static final int GENERIC_ERROR = -1;

    @XmlElement(name = "errorId", nillable = false, required = true)
    public int errorId;

    @XmlElement(name = "errorMsg", required = false)
    public String errorMsg;
```

}

2.10 getAccount(userId, status)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting
?userId=value&status=value

Request format:

String userId, int status

Response Format:

```
@XmlRootElement(name = "accountResponse")
public class AccountResponse extends GenericResponse {

    @XmlElement(name = "user", nillable = false, required = true)
    public List<User> userList;
}
```

2.11 getAccount(status)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting
?status=value

Request format:

int status

Response Format:

```
@XmlRootElement(name = "accountResponse")
public class AccountResponse extends GenericResponse {

    @XmlElement(name = "user", nillable = false, required = true)
    public List<User> userList;
}
```

2.12 getMobileAccount(vehicleId, status)

GET http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/mobile
?vehicleId=value&status=value

Request format:

int vehicleId, int status

Response Format:

```
@XmlRootElement(name = "mobileAccountResponse")
public class MobileAccountResponse extends GenericResponse {

    @XmlElement(name = "mobileUser", nillable = false, required = true)
    public MobileUser user;
}
```

2.13 updateAccount(userId, password, profileId, status, details)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/update

Request format:

```
@XmlType(name = "user")
@XmlRootElement(name = "user")
public class User {

    public final static Integer PENDING_STATUS   = 0;
    public final static Integer CREATED_STATUS  = 1;
    public final static Integer REFUSED_STATUS  = 2;
    public final static Integer DELETED_STATUS   = 3;

    @XmlElement(name = "userId", nillable = false, required = true)
    public String userId;
```

```
    @XmlElement(name = "password", required = false)
    public String password;

    @XmlElement(name = "profileId", required = false)
    public Integer profileId;

    @XmlElement(name = "status", required = false)
    public Integer status;

    @XmlElement(name = "details", required = false)
    public String details;
}
```

Response Format:

```
@XmlType(name = "response")
@XmlRootElement(name = "response")
public class GenericResponse {

    public static final int NO_ERROR = 0;
    public static final int GENERIC_ERROR = -1;

    @XmlElement(name = "errorId", nillable = false, required = true)
    public int errorId;

    @XmlElement(name = "errorMsg", required = false)
    public String errorMsg;
}
```

2.14 checkAccount(userId, password, status)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/check

Request format:

```
@XmlType(name = "user")
@XmlRootElement(name = "user")
public class User {

    public final static Integer PENDING_STATUS   = 0;
    public final static Integer CREATED_STATUS  = 1;
    public final static Integer REFUSED_STATUS  = 2;
    public final static Integer DELETED_STATUS   = 3;

    @XmlElement(name = "userId", nillable = false, required = true)
    public String userId;

    @XmlElement(name = "password", required = false)
    public String password;

    @XmlElement(name = "profileId", required = false)
    public Integer profileId;

    @XmlElement(name = "status", required = false)
    public Integer status;

    @XmlElement(name = "details", required = false)
    public String details;
}
```

Response Format:

```
@XmlType(name = "response")
@XmlRootElement(name = "response")
public class GenericResponse {

    public static final int NO_ERROR = 0;
    public static final int GENERIC_ERROR = -1;
```

```
    @XmlElement(name = "errorId", nillable = false, required = true)
    public int errorId;

    @XmlElement(name = "errorMsg", required = false)
    public String errorMsg;
}
```

2.15 sessionRequest(userId, password, vehicleId)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/sessionRequest

Request format:

```
@XmlRootElement(name = "mobileUser")
public class MobileUser extends User{

    @XmlElement(name = "vid", nillable = false, required = true)
    public Integer vehicleId;
}
```

Response Format:

```
@XmlRootElement(name = "authResponse")
public class AuthResponse extends GenericResponse {

    @XmlElement(name = "seid", nillable = false, required = true)
    public int seid;
}
```

2.15.1 logout(sessionId, vehicleId)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/accounting/logout

Request format:

```
@XmlRootElement(name = "mobileSession")
```

```
public class MobileSession {

    @XmlElement(name = "seid")
    public int sessionId;

    @XmlElement(name = "vid")
    public int vehicleId;
}
```

Response Format:

```
@XmlType(name = "response")
@XmlRootElement(name = "response")
public class GenericResponse {

    public static final int NO_ERROR = 0;
    public static final int GENERIC_ERROR = -1;

    @XmlElement(name = "errorId", nillable = false, required = true)
    public int errorId;

    @XmlElement(name = "errorMsg", required = false)
    public String errorMsg;
}
```

2.16 alarm(sensorId, eventType, time)

POST http://ipermob.isti.cnr.it:8080/EventDB/resources/alarm

Request format:

```
@XmlRootElement(name = "alarm")
public class AlarmMessage {

    @XmlElement(name = "sid")
```

```
    public String sensorId;

    @XmlElement(name = "eventType")
    public int eventType;

@XmlElement(name = "time")
    public long time;
}
```

Response Format:

SUCCESS: “HTTP 200 OK” with an optional String in the body.

FAIL: “HTTP 500 Internal Server Error” with the String containing the error in the body.

2.17 eventParkingSlotStatusChange(List<slotId, state>)

POST http://213.173.101.81:8080/Ipermob-Rest/notify/parkingStatus

Response Format:

```
@XmlRootElement(name = "parkingStatus")
public class NotifyParkingStatus {

    @XmlElement(name = "slot")
    public List<ParkingSlotInfo> parkingSlotList;
}

@XmlRootElement(name = "parkingSlot")
   public class ParkingSlotInfo {

       @XmlElement(name = "slid")
       public int slotId;

       @XmlElement(name = "state")
       public int state;
```

```
}
```

2.18 eventFlowStatusUpdate(period, List<arcId, numVehicles, classA, classB, classC, classD, speed>)

POST http://213.173.101.81:8080/Ipermob-Rest/notify/vehiclesDetection

Response Format:

```
@XmlRootElement(name = "vehicleFlow")
public class NotifyVehicleFlow {

   @XmlElement(name = "period")
   public int period;

   @XmlElement(name = "flow")
   public List<VehicleArcInfo> vehicleFlowList;
}

@XmlRootElement(name = "vehicleArc")
public class VehicleArcInfo {

   @XmlElement(name = "aid")
   public int arcId;

   @XmlElement(name = "numVehicles")
   public int numVehicles;

   @XmlElement(name = "classA")
   public int classA;

   @XmlElement(name = "classB")
   public int classB;
```

```
    @XmlElement(name = "classC")
    public int classC;

    @XmlElement(name = "classD")
    public int classD;

    @XmlElement(name = "speed")
    public int speed;

@XmlElement(name = "time")
    public long time;
  }
```

2.19 eventPositionUpdate(vehicleId, longitude, latitude, arcId, segmentOrder, time)

POST http://213.173.101.81:8080/Ipermob-Rest/notify/positionUpdate

Response Format:

```
@XmlRootElement(name = "positionUpdateNotification")
public class NotifyPositionUpdate {

    @XmlElement(name = "vid")
    public int vid;

    @XmlElement(name = "longitude")
    public float longitude;

    @XmlElement(name = "latitude")
    public float latitude;

    @XmlElement(name = "aid")
    public int arcId;
```

```
@XmlElement(name = "segord")
   public int segmentOrder;

   @XmlElement(name = "time")
   public long time;
}
```

2.20 eventAlarm(time, sensorId, eventType)

POST http://213.173.101.81:8080/Ipermob-Rest/notify/alarm

Response Format:

```
@XmlRootElement(name = "alarm")
public class AlarmMessage {

   @XmlElement(name = "sid")
   public String sensorId;

   @XmlElement(name = "eventType")
   public int eventType;

   @XmlElement(name = "time")
   public long time;
}
```

2.21 getParkingHistory(park)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getParkingHistory

Request format:

```
@XmlRootElement(name = "parkingHistoryParams")
```

```
public class ParkingHistoryParams {

    @XmlElement(name = "parkId")
    public int parkId;

    @XmlElement(name = "startTime")
    public long startTime;

    @XmlElement(name = "endTime")
    public long endTime;
}
```

Response Format:

```
@XmlRootElement(name = "parkingHistory")
public class ParkingHistory {

    @XmlElement(name = "history")
    public List<ParkingStatusElement> historyList;
}

@XmlRootElement(name = "parkingStatusElement")
public class ParkingStatusElement {

    @XmlElement(name = "time")
    public long time;

    @XmlElement(name = "numParkedVehicles")
    public int numParkedVehicles;

}
```

2.22 getCubeList()

GET
http://ipermob2.isti.cnr.it:8080/imob-rest/getCubeList

Response Format:

```
@XmlRootElement(name = "cube")
public class CubeInfo {

   @XmlElement(name = "cube")
   public List<Cube> cubeList;

}

@XmlRootElement(name = "cube")
public class Cube {

    public final static String SLOT="Slot";
    public final static String FLOW="Flow";

    @XmlElement(name = "value")
     public String value;
}
```

2.23 getDimensionsList(cube)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getDimensionsList

Request format:

```
@XmlRootElement(name = "cube")
public class Cube {

    public final static String SLOT="Slot";
    public final static String FLOW="Flow";

    @XmlElement(name = "value")
     public String value;
}
```

Response Format:

```
@XmlRootElement(name = "dimensionInfo")
public class DimensionInfo {

   @XmlElement(name = "dimension")
   public List<Dimension> dimensionList;

}

@XmlRootElement(name = "dimension")
public class Dimension {

   public final static String PARKS = "Parks";
   public final static String TIME ="Time";
   public final static String ARCS = "Arcs";
```

```
    @XmlElement(name = "value")
    public String value;

}
```

2.24 getMeasuresList(cube)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getMeasuresList

Request format:

```
@XmlRootElement(name = "cube")
public class Cube {

    public final static String SLOT="Slot";

    public final static String FLOW="Flow";

    @XmlElement(name = "value")
     public String value;

}
```

Response Format:

```
@XmlRootElement(name = "measureInfo")
public class MeasureInfo {

   @XmlElement(name = "measure")
   public List<Measure> measureList;
```

```
}

@XmlRootElement(name = "measure")
public class Measure {

    public final static String NUM_VEHICLES = "Num vehicles";
    public final static String MAX_SPEED = "Max speed" ;
    public final static String MIN_SPEED = "Min speed" ;
    public final static String CLASSA = "Vehicles ClassA Rate";
    public final static String CLASSB = "Vehicles ClassB Rate";
    public final static String CLASSC = "Vehicles ClassC Rate";
    public final static String CLASSD = "Vehicles ClassD Rate";
    public final static String SPEED = "Avg Speed";
    public final static String SLOT_OCCUPANCY = "Temporal Slots Occupancy";
    public final static String SLOT_OCCUPANCY_FACTOR="Occupancy factor";
    public final static String EVER_BEEN_OCCUPIED= "Has ever been occupied";
    public final static String AVG_PARKING_TIME="Avg Parking Time";

    @XmlElement(name = "value")
      public String value;
}
```

2.25 getLevelsForDimension(dimension)

GET
http://ipermob2.isti.cnr.it:8080/imob-rest/getLevelsForDimension?dimension=value

Request format:
String dimension

Response Format:

```
@XmlRootElement(name = "levelInfo")
public class LevelInfo {

  @XmlElement(name = "level")
  public List<Level> levelList;

}

@XmlRootElement(name = "level")
public class Level {

  public final static String YEAR = "Year";
  public final static String MONTH = "Month";
  public final static String DAY = "Day";
  public final static String HOUR = "Hour";
  public final static String MINUTES = "Minutes";
  public final static String PARK = "Park";
  public final static String SLOT = "Slot";
  public final static String ARC = "Arc";
  public final static String FLOW = "Flow";

  @XmlElement(name = "value")
  public String value;

}
```

2.26 getPivotTable(PivotParams p)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getPivotTable

Request format:

```
@XmlRootElement(name = "pivotParams")
public class PivotParams {

  @XmlElement(name = "dim1")
  public String dim1;

  @XmlElement(name = "dimTime")
  public String dimTime;

  @XmlElement(name = "measure")
  public String measure;

  @XmlElement(name = "cube")
  public String cube;

}
```

Response Format:

```
@XmlRootElement(name = "pivotInfo")
public class PivotInfo {

  @XmlElement(name = "infoString")
  public String cube;

}
```

2.27 getArcHistory(arc)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getArcHistory

Request format:

```
@XmlRootElement(name = "arcHistoryParams")
public class ArcHistoryParams {

    @XmlElement(name = "aid")
    public int arcId;

    @XmlElement(name = "startTime")
    public long startTime;

    @XmlElement(name = "endTime")
    public long endTime;
}
```

Response Format:

```
@XmlRootElement(name = "arcHistory")
public class ArcHistory {

    @XmlElement(name = "history")
    public List< VehicleArcInfo > historyList;
}

@XmlRootElement(name = "vehicleArc")
public class VehicleArcInfo {

    @XmlElement(name = "time")
     public long time;

     @XmlElement(name = "aid")
      public int arcId;

     @XmlElement(name = "numVehicles")
      public int numVehicles;
```

```
    @XmlElement(name = "classA")
    public int classA;

    @XmlElement(name = "classB")
    public int classB;

    @XmlElement(name = "classC")
    public int classC;

    @XmlElement(name = "classD")
    public int classD;

    @XmlElement(name = "speed")
    public int speed;

}
```

2.28 getGraph(time)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getGraph

Request format:

```
@XmlRootElement(name = "basicParams")
public class BasicParams {

   @XmlElement(name = "time")
   public long time;

}
```

Response Format:

```
@XmlRootElement(name = "getGraphResponse")
public class GetGraphResponse {

    @XmlElement(name = "archInfo")
    public List<ArchInfo> archInfoList;

}

@XmlRootElement(name = "archInfo")
public class ArchInfo {

    @XmlElement(name = "aid")        // objectid
    public int archId;

    @XmlElement(name = "origNodeId")      // a
    public int origNodeId;

    @XmlElement(name = "destNodeId")     // b
    public int destNodeId;

    @XmlElement(name = "scenId")           // cod_scen
    public String  scenId;

}
```

2.29 getZones(time)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getZones

Request format:

```
@XmlRootElement(name = "basicParams")
public class BasicParams {

    @XmlElement(name = "time")
    public long time;

}
```

Response Format:

```
@XmlRootElement(name = "getZonesResponse")
public class GetZonesResponse {

    @XmlElement(name = "zoneInfo")
    public List<ZoneInfo> zoneInfoList;

}

@XmlRootElement(name = "zoneInfo")
public class ZoneInfo {

    @XmlElement(name = "zoneId")
    // id
    public int zoneId;

    @XmlElement(name = "nodeId")
    // node
    public int nodeId;

    @XmlElement(name = "addetti")
    public double addetti;

    @XmlElement(name = "popolazione")
```

```
    public double popolazione;

    @XmlElement(name = "geometry")
    public String geometry;

}
```

2.30 getODMatrix(time)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getODMatrix

Request format:

```
@XmlRootElement(name = "basicParams")
public class BasicParams {

    @XmlElement(name = "time")
    public long time;

}
```

Response Format:

```
@XmlRootElement(name = "getODMatrixResponse")
public class GetODMatrixResponse {

    @XmlElement(name = "odmatrixInfo")
    public List<ODMatrixInfo> odmatrixList;

}
```

```
@XmlRootElement(name = "odmatrixInfo")
public class ODMatrixInfo {

    @XmlElement(name = "id")
    public int id;

    @XmlElement(name = "origZoneId")        // o
    public int origZoneId;

    @XmlElement(name = "destZoneId")        // d
    public int destZoneId;

    @XmlElement(name = "flowNum")           // flusso
    public float flowNum;

    @XmlElement(name = "codiceMat")
    public int codiceMat;

}
```

2.31 getODMatrixCatalog()

GET
http://ipermob2.isti.cnr.it:8080/imob-rest/getODMatrixCatalog

Response Format:

```
@XmlRootElement(name = "getODMatrixCatalogResponse")
public class GetODMatrixCatalogResponse {

    @XmlElement(name = "codiceMats")
    public int[] codiceMats;
```

```
}
```

getArcFlow(string)

POST

http://ipermob2.isti.cnr.it:8080/imob-rest/getArcFlow

Request format:

```
@XmlRootElement(name = "getArcFlowRequest")
public class GetArcFlowRequest {

    @XmlElement(name = "scenId")
    public String scenId;

}
```

Response Format:

```
@XmlRootElement(name = "getArcFlowResponse")
public class GetArcFlowResponse {

    @XmlElement(name = "arcFlowInfo")
    public List<ArcFlowInfo> arcFlowInfoList;

}

@XmlRootElement(name = "arcFlowInfo")
public class ArcFlowInfo {

    @XmlElement(name = "archId")          // objectid
    public int archId;

    @XmlElement(name = "origNodeId")      // a
```

```
    public int origNodeId;

    @XmlElement(name = "destNodeId")        // b
    public int destNodeId;

    @XmlElement(name = "scenId")           // cod_scen
    public String  scenId;

    @XmlElement(name = "name")             // name
    public String name;

    @XmlElement(name = "flow")             // flow
    public int flow;

}
```

2.32 getArcTravelTime(string)

POST
http://ipermob2.isti.cnr.it:8080/imob-rest/getArcTravelTime

Request format:

```
@XmlRootElement(name = "getArcTravelTimeRequest")
public class GetArcTravelTimeRequest {

    @XmlElement(name = "scenId")
    public String scenId;

}
```

Response Format:

```
@XmlRootElement(name = "getArcTravelTimeResponse")
public class GetArcTravelTimeResponse {

    @XmlElement(name = "arcTravelTimeInfo")
    public List<ArcTravelTimeInfo> arcTravelTimeInfoList;

}

@XmlRootElement(name = "arcTravelTimeInfo")
public class ArcTravelTimeInfo {

    @XmlElement(name = "archId")        // objectid
    public int archId;

    @XmlElement(name = "origNodeId")    // a
    public int origNodeId;

    @XmlElement(name = "destNodeId")    // b
    public int destNodeId;

    @XmlElement(name = "scenId")        // cod_scen
    public String scenId;

    @XmlElement(name = "name")          // name
    public String name;

    @XmlElement(name = "minutes")       // minutes
    public double minutes;

}
```

2.33 getGraphCatalog()

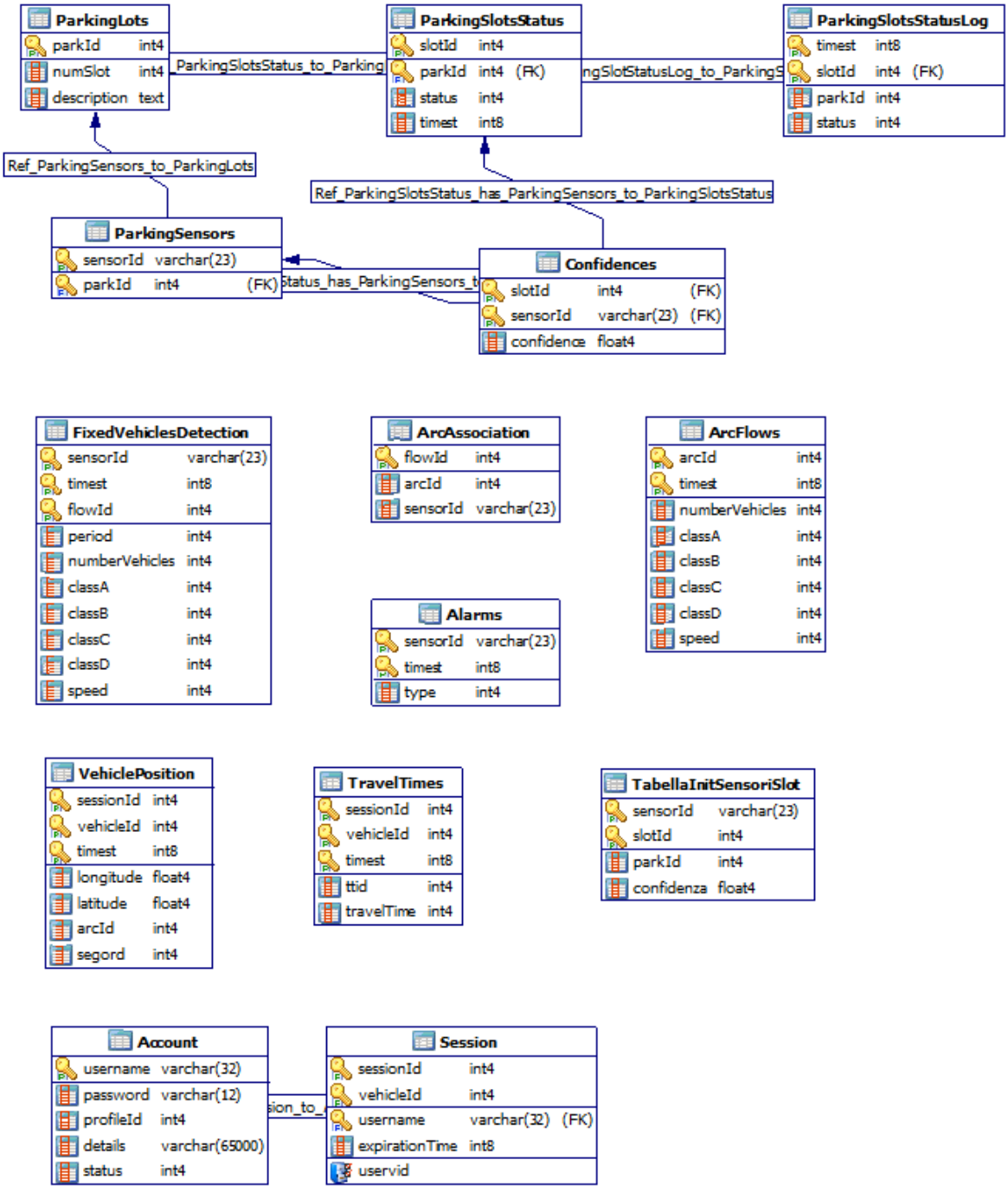
GET
http://ipermob2.isti.cnr.it:8080/imob-rest/getGraphCatalog

Response Format:

```
@XmlRootElement(name = "graphCatalogResponse")
public class GraphCatalogResponse {

    @XmlElement(name = "graphCatalogInfo")
    public List<GraphCatalogInfo> graphCatalogInfoList;

}

@XmlRootElement(name = "graphCatalogInfo")

public class GraphCatalogInfo {

   @XmlElement(name = "codScenario")

   public String codScenario;

}
```

3 APPENDICE - SCHEMI DEI DATABASE

3.1 Schema dell'Event Database



3.2 Schema dell'Off Line Database

Lo schema dell'Off Line Database è stato rilasciato nel documento *Data Warehouse Ipermob - Schema logico e implementazione*.

3.3 Schema dello Scenario & Models Database

Lo schema dello Scenario & Models Database è stato rilasciato nel documento *deliverable 2 - attività 4.1 costruzione degli scenari per la piattaforma modellistica: modello Micro.*