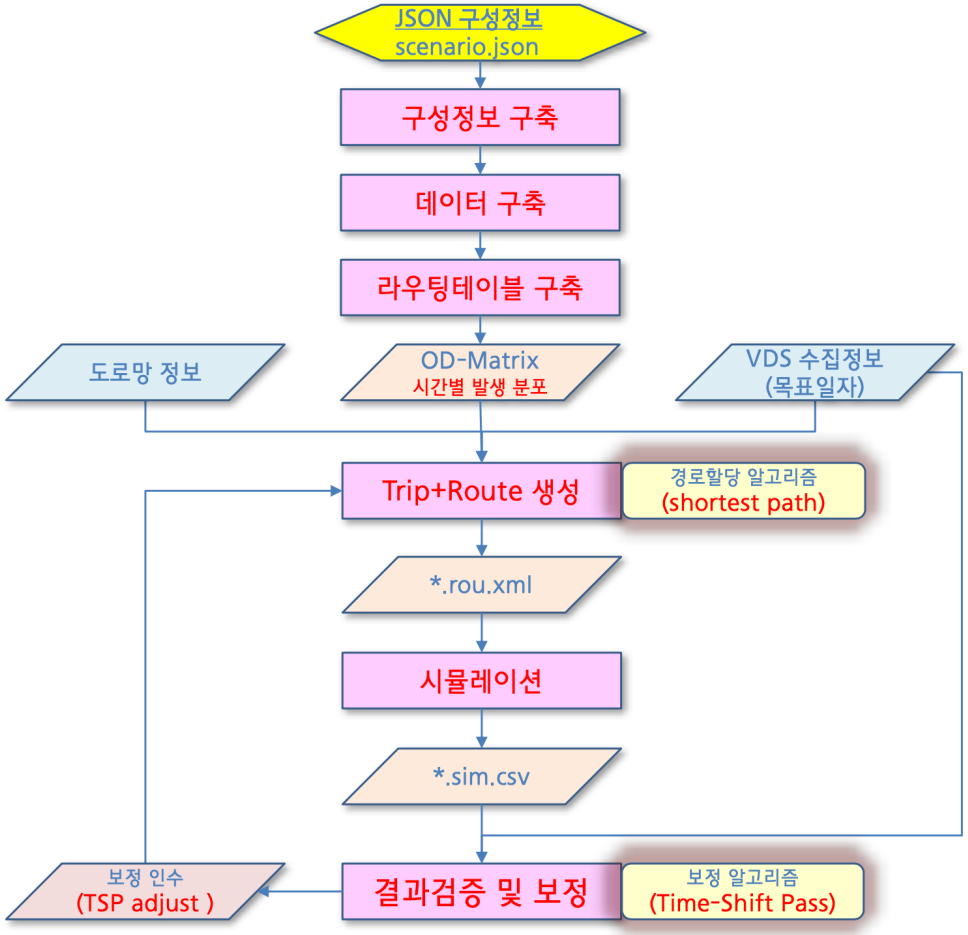
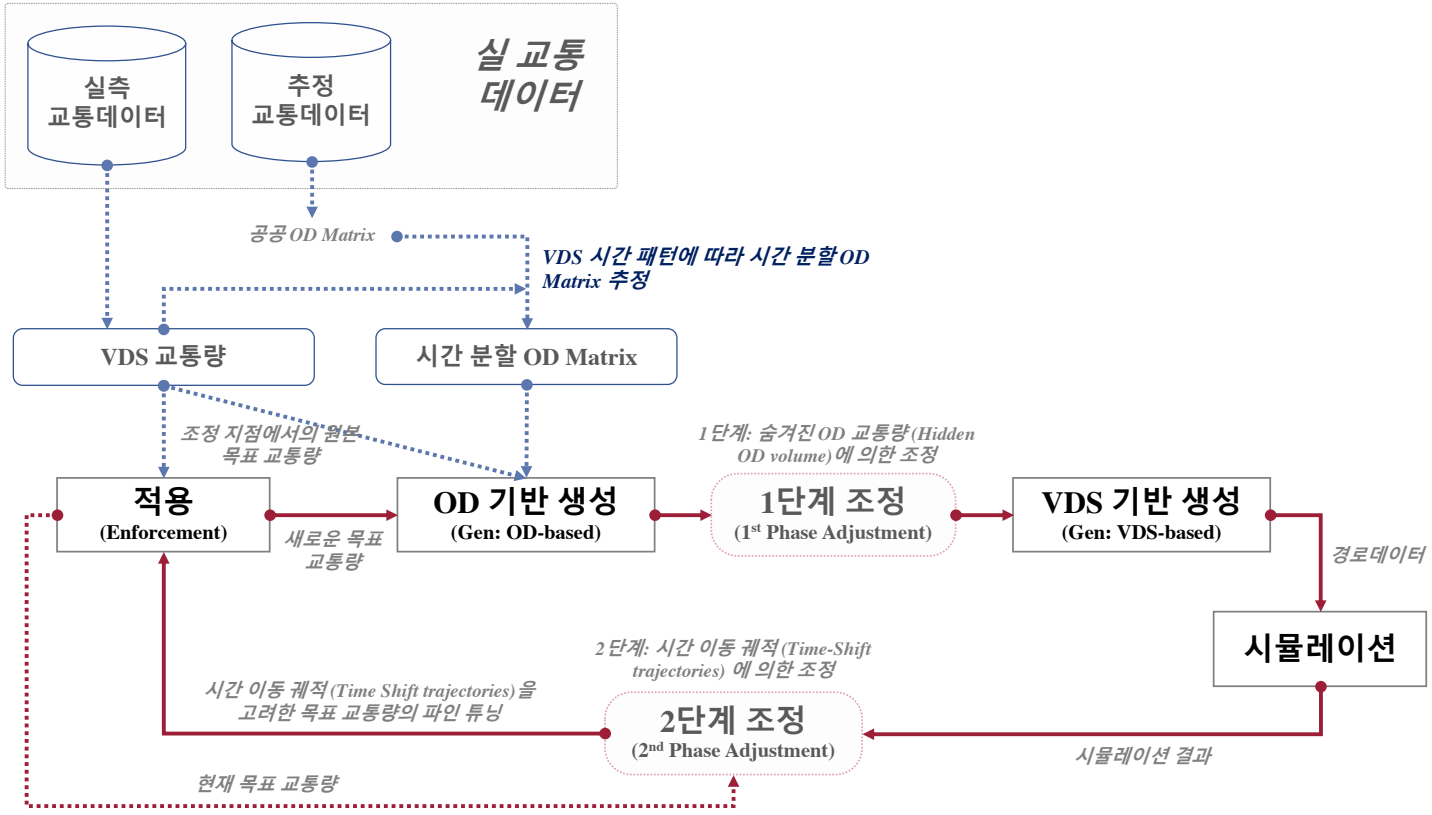
수요 할당 도구 개발 계획

기 개발

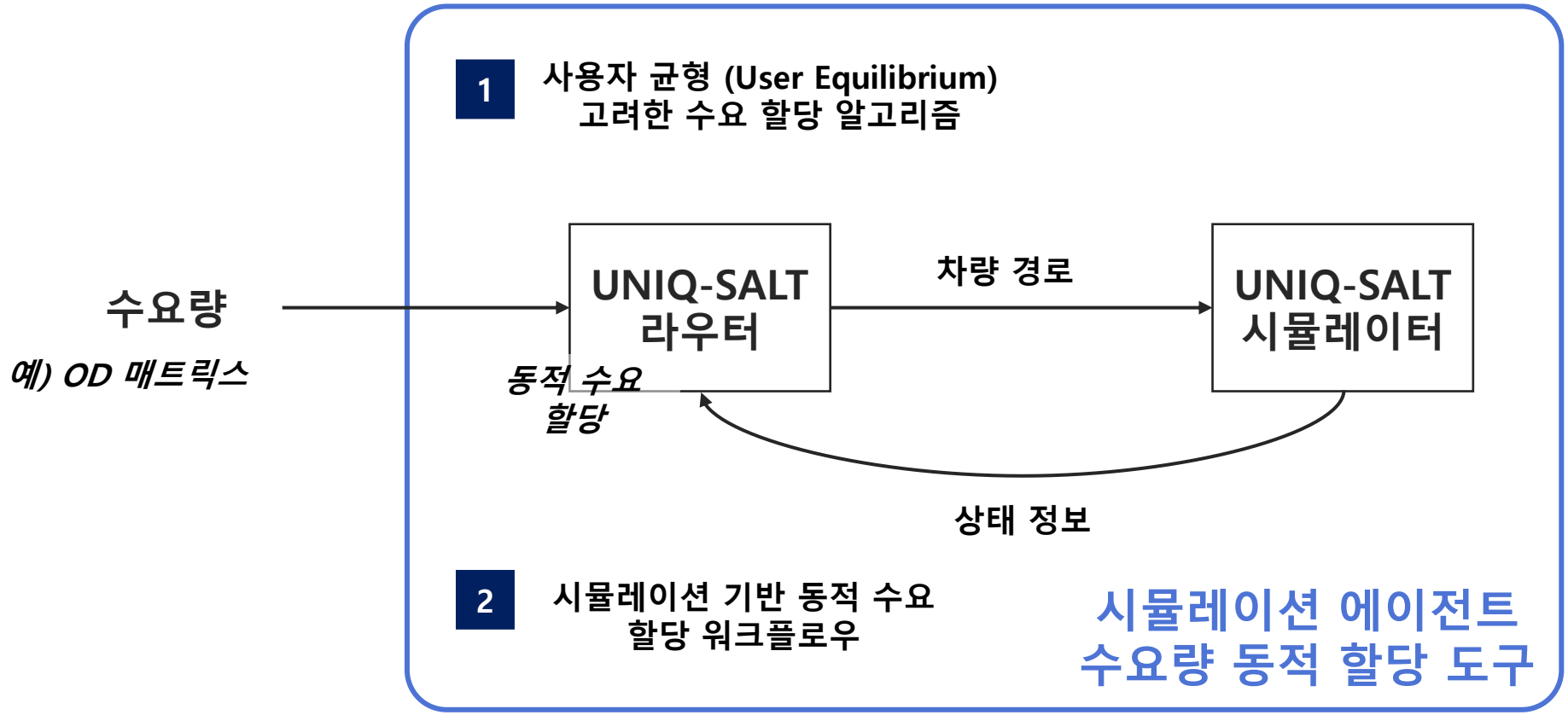
○ 최단 경로 알고리즘 기반 경로 할당 도구 개발

- ▸ KTDB OD matrix 활용 + VDS 지점 통과량 보정 활용





용역 범위 개념도


1
사용자 균형 (User Equilibrium)
고려한 수요 할당 알고리즘
수요량
예) OD 매트릭스
UNIQ-SALT
라우터
차량 경로
UNIQ-SALT
시뮬레이터
동적 수요
할당
상태 정보
2
시뮬레이션 기반 동적 수요
할당 워크플로우
시뮬레이션 에이전트
수요량 동적 할당 도구

용역 범위

○ 사용자 균형을 고려한 시뮬레이션 에이전트 수요 데이터 할당 알고리즘 구현

- ▸ 교통 시뮬레이션 연계를 통한, 사용자 균형을 고려한 수요 할당 알고리즘 분석
- ▸ UNIQ-SALT 교통 시뮬레이터를 활용하여, 사용자 균형을 고려한 수요 할당 알고리즘 구현

※ **사용자 균형을 고려한 수요 할당 알고리즘은 SUMO의 Traffic Assignment (휴리스틱 반복 할당 방식) 를 참고**
※ 반복 할당 방식 외 동적 할당 기법은 필요시 추가적으로 구현될 수 있고, 이는 상호 협의하에 조율할 수 있음

○ 교통 시뮬레이션 기반 시간별 교통 상황 변화에 따른 동적 할당 워크플로우 설계 및 구현

- ▸ 시뮬레이션 에이전트 수요량을 동적으로 할당하기 위한, 동적 사용자 균형 (Dynamic User Equilibrium, DUE)을 고려한 경로 할당 알고리즘 구현
- ▸ 교통 시뮬레이션 연계를 통해, 시간별로 변화하는 교통 상황을 반영하는 동적 사용자 균형을 고려한 경로 할당 알고리즘 기반 수요 할당 워크플로우 설계

※ **경로 탐색을 위해,** UNIQ-SALT 라우터의 기본 라우팅 알고리즘인 최단 거리 알고리즘 외 **최단 시간 알고리즘 적용**
※ 동적 사용자 균형을 고려한 경로 선택을 위한 알고리즘은 SUMO의 DUA(Dynamic User Assignment) 방식에서 사용된 경로 선택 알고리즘 참고하여, 1개 이상 적용

○ 시뮬레이션 에이전트 수요량 동적 할당 위한 사용자 UI 제공

- ▸ 사용자가 수요량 데이터에 대한 동적 할당을 통해 시뮬레이션 에이전트 수요 데이터를 생성하여 그 결과를 확인할 수 있는 사용자 UI를 제공

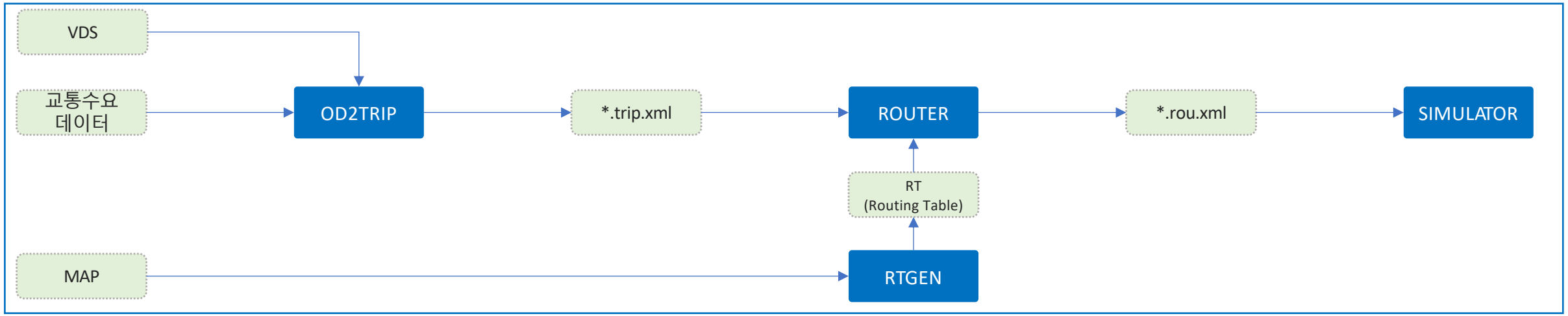
[참고] SUMO DUA Router

- https://sumo.dlr.de/docs/Demand/Dynamic_User_Assignment.html
  - This script tries to calculate a user equilibrium, that is, it tries to find a route for each vehicle (each trip from the trip-file above) such that each vehicle cannot reduce its travel cost (usually the travel time) by using a different route. It does so iteratively (hence the name) by
    - calling duarouter to route the vehicles in a network with the last known edge costs (starting with empty-network travel times)
    - calling sumo to simulate "real" travel times result from the calculated routes. The result edge costs are used in the net routing step.

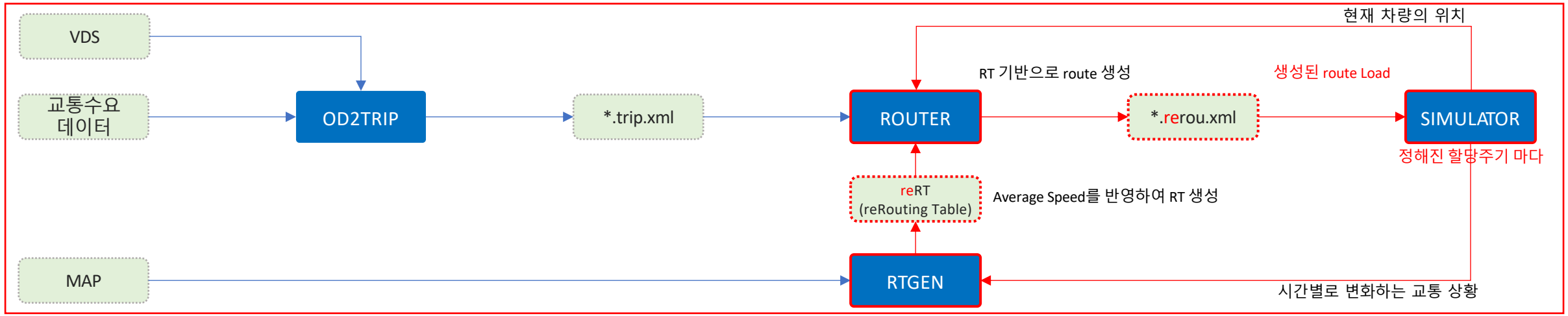
→ Python 코드로 제공 (dualterate.py)

수행 방안 – 개요

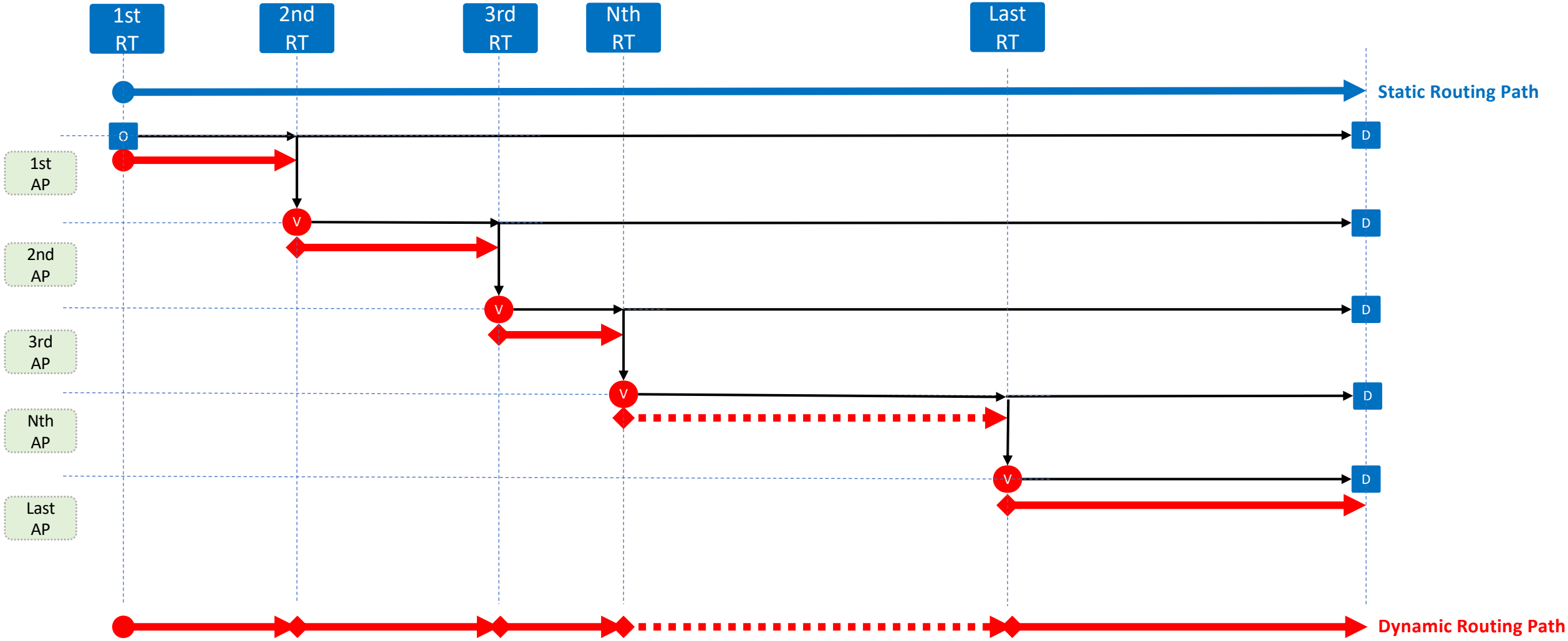
Static Routing



Dynamic Routing



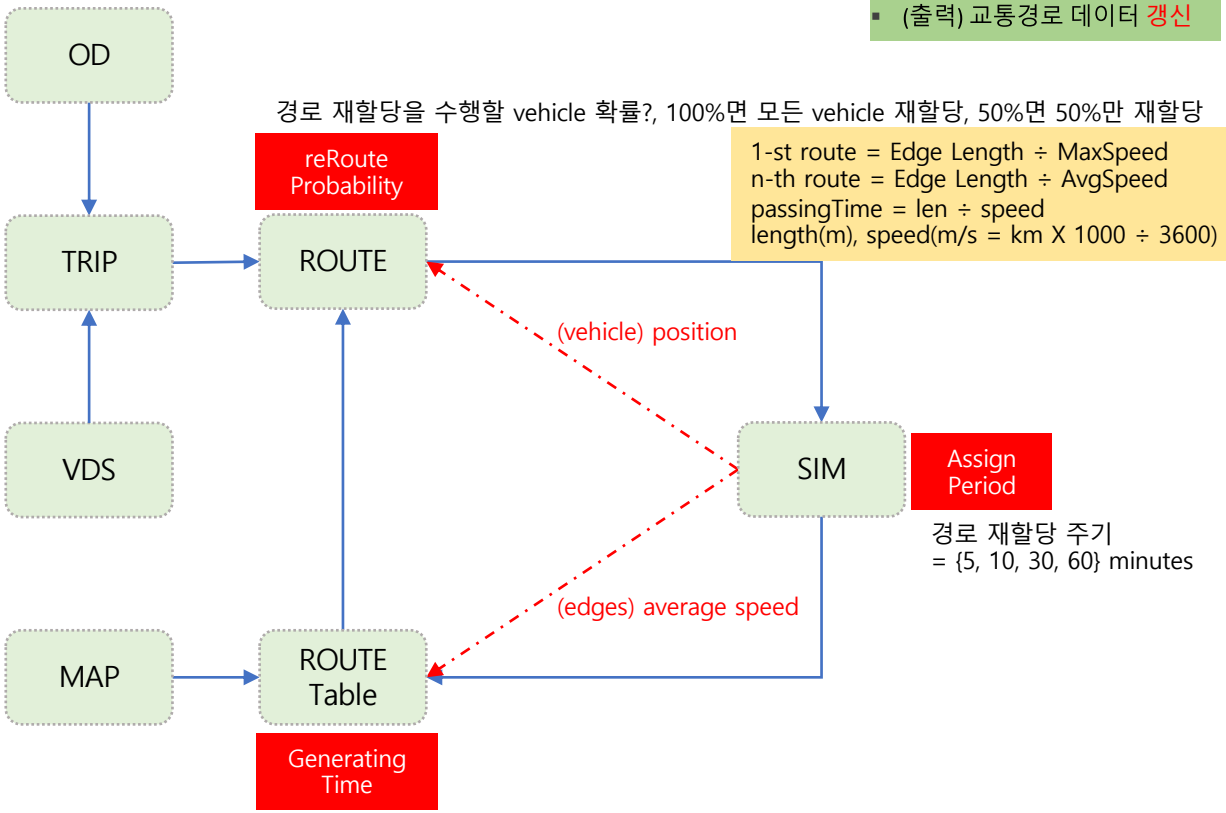
수행 방안 – 개요


1st RT
2nd RT
3rd RT
Nth RT
Last RT
Static Routing Path
O
D
1st AP
V
2nd AP
3rd AP
Nth AP
Last AP
Dynamic Routing Path

개발 시나리오

DUE(Dynamic User Equilibrium) = shortest time algorithm + reRoute Probability
DUA(Dynamic User Assignment) = oneShot-assignment

- (입력) 차량ID, 현재 경로 위치
- (입력) 도로ID의 평균속도,길이
- (처리) 동적 경로할당
- (출력) 교통경로 데이터 갱신



경로 재할당을 수행할 vehicle 확률?, 100%면 모든 vehicle 재할당, 50%면 50%만 재할당

1-st route = Edge Length ÷ MaxSpeed
n-th route = Edge Length ÷ AvgSpeed
passingTime = len ÷ speed
length(m), speed(m/s = km X 1000 ÷ 3600)

경로 재할당 주기
= {5, 10, 30, 60} minutes

라우팅 테이블을 생성하는데, 얼마나 시간이 소요되나 ?

① **Trip(*.trip.xml) 생성**
- { OD + VDS } → TRIP 생성.
- OD lack traffic을 보정하기 위하여 VDS 실측 정보 활용 필요

② **1-st RT(Routing Table) 생성**
- { MAP } → RT(Routing Table) 생성 : 1-st RT weight = Edge Length ÷ Max Speed

③ **1-st Route (*.rou.xml) 생성**
- { TRIP + RT } → 경로 생성 : { O , ...Route... , D }

④ **시뮬레이션 수행**
- *.rou.xml → SIM : 시뮬레이션 수행
- 보정 정보 생성 :

  Assign Period(5, 10, 30, 60) minutes 마다 { Edge Average Speed + vehicle Position } 참조
- *Edge Average Speed : *-PeriodicOutput.csv =*

  { intervalbegin,intervalend,roadID,VehPassed,AverageSpeed,AverageDensity,WaitingQLength,WaitingTime,SumTravelLength,SumTravelTime }
- *vehicle Position : *.vpos.csv = { vehicleID,roadID }*

⑤ **n-th rRT(re-Routing Table) 생성**
- n-th RT weight = Edge Length ÷ Average Speed
- reRoute Probabilty : 경로 재할당을 수행할 vehicle 재할당 확률 { 100, 70, 50, 30, 0 } %

⑥ **Re-Route 생성**
- reRT + vPos → 경로 재생성 : { O , ..., V, ...reRoute... , D }
- vehicle Position을 Original로 하여 Destination까지 경로 재할당

⑦ **시뮬레이션 수행**
- *.rou.xml → SIM : 시뮬레이션 수행
- Route 파일을 reRoad하여 시뮬레이션에 적용

이슈 사항 – 1

1. RT 재생성 소요 시간
   - RT 생성 계산시간은 대상 링크수에 대하여 지수적으로 증가
   - RT생성시간이 10분이고, 재할당 주기가 30분인 경우, 10분 X 1440분 / 30분 = 480분 = 8시간
   - 만약, 전체 Edge가 10,000개 이면 10,000 X 10,000 = 100,000,000 번 계산
   - 감내 할 수 있는 최대 시뮬레이션 시간 고려 필요
   - ??? maxSpeed에 대비하여 avgSpeed가 특정비율(1.0 ~ 0.0) 이하인 EDGE만 RT 재생성 ???
   - ??? DONG_CD를 기준으로 RT_Local를 여러 개 생성하고, DONG_CD의 경계에 위치하는 EDGE만으로 새로운 RT_Global를 생성하여 2단계(RT_Global → RT_Local) 라우팅???

2. 동적 경로 재할당 주기
   - 예, Assign Period(5, 10, 30, 60) minutes
   - 주기가 짧으면 동적 할당의 의미는 커지나, 처리시간이 길어짐.
   - 주기가 길면 동적 할당의 의미는 작아지고, 처리시간은 짧아짐.
   - SUMO에서 30분 기본.

3. 경로 재할당을 수행할 vehicle 확률
   - 예, vehicle 재할당 확률 { 100, 70, 50, 30, 0 } %
   - 재할당 확률이 크면, 모든 차량이 특정(빠른 AvgSpeed) EDGE로 집중될 수 있음.
   - 재할당 확률이 작으면, Dynamic 의미가 작아짐.
   - SUMO에서 50% 기본.
   - ???또는, 현재 Route시간 대비 갱신 Route 시간 차이가 임계값을 상회하는 경우만 라우팅 재할당 ???

이슈 사항 – 2

4. Assign Period에 대한 SC 운용 방법
   - SC-> doSimulationStep(currentStep) 함수를 Assign_Period 적용 필요

5. 재할당 Route* route에 대한 reload 방법
   - SC->VehicleManager->VehicleInterface->Route->reload(Route* route) 처가 정의 필요

6. 시뮬레이션 정보 제공 방법
   - Average Speed : float avgSpeed = Link->getAverageSpeed()
   - Vehicle position : `Link* link = vehicleIF->getCurrentLink();`

7. 프로그램 운용(변경) 방안 고려시, router는 시뮬레이션과 하나라고 생각한 방안 필요
   - Simulator가 main()이고 내부적으로 router()를 사용

8. 1-st ~ n-th 라우팅 경로(이력) 저장 필요성
   - 검증을 위해 필요할 듯…
   - 1-st & last Path 저장

이슈 사항

```
#include <utils/config.h>
#include <utils/etc.h>
#include "Controller/SimulationController.h"

typedef std::chrono::high_resolution_clock Clock;

using namespace std;
using namespace SALT;

int main_template(int argc, char **argv){...}

int main(int argc, char **argv){
        // '''Procedure'''
        // 1. locate input file
        // 2. configure simulation setting (load input)
        // 3. do simulation step

        // ------------------------------------------------------------------------
        // @ Locate input file
        string scenarioFile="";
        if(argc>=2){
         scenarioFile = string(argv[1]);
        }else{
         cerr << "Specify scenario file" << endl;
        return -1;
        }

        cout << "[SALT Simulator] "<<endl;

        // ------------------------------------------------------------------------
        // @ Configure simulation controller (load input)
        cout<<"-----Configuration File : "<<scenarioFile << endl;
        SimulationController* SC = new SimulationController();
        Result configResult = SC->configureSimulation(scenarioFile);
        if(configResult==FAILURE){
         cerr<<"-----Configuration Failure-----"<<endl;
         return -1;
        }else{
         cout<<"-----Configuration Done-----"<<endl;
        }

        SC->printSimulationSetting();

        // ------------------------------------------------------------------------
        // @ do simulation loop
        std::cout << "[Simulation Start]" << std::endl;
        auto timeStart = Clock::now();
        int NUMBER_PRINT = 20; // simulation progress states will be printed (NUMBER_PRINT) times // 20 -> each 5%

        // repeat to call SimulationController->doSimulationStep(currentStep)
        while(!SC->checkEnd()) {
         // get current step + update status of vehicles, traffic signal and event
         SALTTime currentStep = SC->getCurrentStep();
         SC->doSimulationStep(currentStep); // doSimulationStep will increase SC::currentStep

         // (optional) print
         SC->printStep(currentStep,NUMBER_PRINT);
        }

        auto timeEnd = Clock::now();
        auto totalTime = chrono::duration_cast<std::chrono::seconds>(timeEnd - timeStart).count();
        cout<<"[Simulation End]"<<endl;
        cout << "Elapsed Time: "<< totalTime << " seconds" << std::endl;

        delete SC;
        SC = nullptr;

        return 0;
}
```

이슈 사항

```
// ------------------------------------------------------------------------
// @ do simulation loop
std::cout << "[Simulation Start]" << std::endl;
auto timeStart = Clock::now();

// int NUMBER_PRINT = 20; // simulation progress states will be printed (NUMBER_PRINT) times // 20 -> each 5%
float NUMBER_PRINT = 8.33333; // simulation progress states will be printed (NUMBER_PRINT) times // 8.33333333333...  -> 30 minutes

// repeat to call SimulationController->doSimulationStep(currentStep)
while(!SC->checkEnd()) {
    // get current step + update status of vehicles, traffic signal and event
    SALTTime currentStep = SC->getCurrentStep();

    SC->loadRoute(Route* route);  // 현재 상태에서 재할당된 Route 적재, VehicleInterface class 에서 Route Class에 접근할수 없는 상태
    SC->doSimulationStep(currentStep); //ASSIGN_PERIOD // doSimulationStep will increase SC::currentStep
    SC->printStep(currentStep,NUMBER_PRINT);

     Route* route = reRouter(); // average_speed , vehicle_position
    // (optional) print
}
```

```
class SimulationController {
public:
    const std::vector<Link*>& getLinkList() const;

    VehicleManager* myVehicleManager=NULL;
    VehicleManager* loadVehicles();

}
```

```
// TEST
list<VehicleInterface*> StandbyList = SC->myVehicleManager->getStandbyList();
set<VehicleInterface*> RunningSet = SC->myVehicleManager->getRunningSet();
VehicleInterface* vehicleIF = SC->myVehicleManager->getVehicle("1");
string vehicleID = vehicleIF->getMyName();
string RouteString = vehicleIF->getRouteString();
Link* link = vehicleIF->getCurrentLink();
float avgspeed = link->getAverageSpeed();
```

Average speed

```
class Link {
    public:
    float getAverageSpeed() const {
        return myAverageSpeed;
    }
}
```

Vehicle position

```
class VehicleManager {
public:
    VehicleInterface* getVehicle(const std::string& vehicleID);

    list<VehicleInterface*> getStandbyList();
    set<VehicleInterface*> getRunningSet();
}
```

Route Class에 접근할 수 없는 상태

```
class VehicleInterface {
public:
    ID& getMyName()  {return myID; }
    Link* getCurrentLink() const;
    string getRouteString() const;
    Route* getRoute() {return myRoute; };
}
```

재할당된 Route 무결성 적재 필요

```
class Route {
public:
    reloadRoute(string reroute); //정의필요
private:
    vector<string> myGlobalRoute;
    string myGlobalRouteString;
}
```

이슈 사항

```
/* VehicleInterface.h */

#ifndef VEHICLEINTERFACE_H_
#define VEHICLEINTERFACE_H_

#include <utils/config.h>
#include <iostream>
#include <list>
#include <map>
#include <string>
#include <utility>


namespace SALT{
class Route;
class CellInterface;
class Link;
class TrafficSignal;

class VehicleInterface {
public:
    VehicleInterface();
    VehicleInterface(Route* _route, string _departTime, int _routeIndex,  ID _ID);
    VehicleInterface(Route* _route, string _departTime, int _routeIndex,  ID _ID,
VehicleClass _class);
    virtual ~VehicleInterface();

…

    string getRouteString() const;

…
```

```
protected:
    //  (constant) fixed values during simulation
    //public:
    ID myID;
    //string myType="";
    VehicleClass myClass = PASSENGER; // VehicleClass: Passenger or Bus
    string myShape="";
    // TODO - input file doesn't have the below info(length, max speed)
    Length myLength = 5.0; // Passenger: 5m, Bus: 11m

    // (constant) fixed values during sub-simulation
    Route* myRoute=0;
    string myRouteString="";

    SALTTime myDepartTime=-1;
    int myRouteDepartingLinkIndex = -1;
    int myRouteDepartingSectionIndex = -1;
    int myRouteDepartingLaneIndex = -1;


    // (variable) status - position, speed
    int myLinkIndex = -1; // [0,myRoute.size()]
    int myCellIndex = -1; // [0,myCellSeries.size()] //myCellSeries.size()<=link's #sections
    list<pair<int,int>> myCellSeries;
    // Speed mySpeed=-1;
    Speed mySpeed = 0;


    list<Link*> mySubNextLinkListCache; // [ current link, ... next valid link]
//  tuple<CellInterface*, CellInterface*> myNextValidCellCache; // <current cell, next valid
cell>
    CellInterface* myCurrentCell=nullptr;
    CellInterface* myNextValidCell=nullptr;

    map<int,map<int,tuple<SALTTime,SALTTime,SALTTime>>> myPassingInfo;
    // map[link_index][cell_index] = <entered time, time to start waiting, leaved time>
};
ostream& operator << (ostream& strm, VehicleInterface& obj);
}
#endif /* VEHICLEINTERFACE_H_ */
```

이슈 사항

```
/* Route.h */

#ifndef OBJECTT_ROUTE_H_
#define OBJECTT_ROUTE_H_

#include <string>
#include <vector>

#include <utils/config.h>
////////////////////////////////////////////////////////
// XXX - consideration
// Should Route class have departing position(link, lane, offset) of vehicle?
////////////////////////////////////////////////////////

namespace SALT{

        class NetworkManager;
        class Link;

        class Route {
        public:
            Route();
            Route(string _linkListString, NetworkManager* networkManagerToRef,
                  int _localOffsetFromGlobal=0, string _departLane="");
            Route(vector<Link*> _linkList);

            ~Route();

            Link* getLink(int index);

            //  Link* getDepartingLink();
            Link* getNextLink(Link* link);
            Link* getNextLink(int from_index);

            //Link* getNextValidLink(Link* link);
            Link* getNextValidLink(int from_index);
            int getOffsetToNextValidLink(int from_index);
            bool validateConnectivity();
            bool isAllInvalidLink();
            bool isSimulatable(){ return flagSimulatable; }
            string getGlobalRouteString() const { return myGlobalRouteString;}
            // @ brief: compute next localOffsetFromGlobal used in the next worker.
            // return ( localOffsetFromGlobal + #(localRoute's valid link) )
            // this value is used in the next worker that receive the vehicle following this route.
            void setNextLocalRouteOFfset(int nextOffset){
                nextLocalRouteOffset=nextOffset;
            }
            int getNextLocalRouteOffset() const { return nextLocalRouteOffset;}

            // determine when to call popVeh()
            // if last local route and last link -> call popVeh()
            bool isLastLocalRoute() { return flagLastLocalRoute;}
            void setLastLocalRoute(){   flagLastLocalRoute = true;  }


            // @brief: given global route and the local offset, build localRoute
            // @return: success or failure to build local route
            Result buildLocalRoute(string _linkListString, NetworkManager* networkManagerToRef);
            string getRemainingRoute(int startIndexOfRemainingRoute);

            Link* getLastLinkOnLocalRoute();
        //private:
            // [fixed during total simulation]
            vector<string> myGlobalRoute;
            string myGlobalRouteString;

            // [fixed during sub simulation, but varying per sub simulation]
            int myLocalRouteOffset=0;
            //  // a sub-simulation doesn't have Link* belonging to other sub-simulation!!
            vector<Link*> myLocalRoute; // <- defined by myGlobalRoute and localOffsetFromGlobal
            int nextLocalRouteOffset=-1; // := myLocalOFfsetFromGlobal + |myLocalRoute|
            bool flagLastLocalRoute = false;

            // XXX - issue
            int myDepartLane=0;

            bool flagSimulatable=true;

            // startingOffset is deprecated. not gonna use the 'offset of a single link'
            // XXX - myStartingIndex is deprecated, because Vehicle has its current position

            static RandomGenerator* myRandomGenerator;
        };
}
#endif /* OBJECTT_ROUTE_H_ */
```