Contents

1. introduction

1.1. about

sumo : Simulation of Urban MObility

microscopic, multimodal traffic simulation

traffic demand : 교통 수요

network : 교통 네트워크, 사실상 지도의 의미도 포함하고 있는 걸로 보임.

1.2. features

- simulation
  - space-continuous & time-discrete vehicle movement
  - different vehicle types
  - multi-lane streets with lane changing
  - different right-of-way rules, traffic lights
  - 빠른 opengl gui
  - 10000개의 간선 네트워크 다룸 (거리)
  - 1GHz 머신에서 초당 100000 차량 업데이트 가능
  - 런타임에 다른 앱과 상호운용성
  - network-wide, edge-based, vehicle-based, and detector-based outputs
  - supports person-based inter-modal trips
- network import
  - VISUM, Vissim, Shapefiles, OSM, RoboCup, MATsim, OpenDRIVE, and XML-Descriptions 싹 다 받음.
  - 결측값은 휴리스틱으로 채워짐
- routing
  - microscopic routes : 각 차량은 자신 고유의 경로가 있다.
  - different dynamic user assignment algorithm
- 오직 c++ 표준라이브러리만 썼고, high portable.

1.3. included applications

- sumo : cli 앱.
- sumo-gui
- netconvert : 다양한 포맷 읽어 SUMO-format으로 변환
- netedit : gui 앱
- netgenerate
- duarouter : DUA(dynamic user assignment) 수행. 네트워크 상 가장 빠른 routes 계산
- jtrrouter : Computes routes using junction turning percentages
- dfrouter : Computes routes from induction loop measurements
- marouter : Performs macroscopic assignment
- od2trips : Decomposes O/D-matrices into single vehicle trips
- polyconvert : POI와 다각형을 받아서 sumo-gui에서 시각화 되는 형식과 description으로 변환
- activitygen : 모델링된 인구의 이동성 요구 사항을 기반으로, 수요(demand) 생성
- emissionsMap - 배기가스 배출 관련
- emissionsDrivingCycle : Calculates emission values based on a given driving cycle
- Additional Tools

1.4. software design criteria

크게 두 디자인 목표가 있다: 속도와 이식성

2. basic usage

2.1. notation

이 문서 상에서의 표기법에 대하여.

괄호 '[', ']'는 옵셔널 정보에 대한 표기법.

괄호 '<', '>'는 변수에 대한 표기법.

2.1.1. cli

command line option name

** < their value> **

2.1.2. XML

`XML element and attributes`

`<their value>`

2.1.3. referenced data types

- <BOOL> :
- <INT>
- <UINT>
- <FLOAT>
- <TIME> : 초 단위
- <STRING>
- <STRING[]>
- <ID>
- <FILE> or <FILENAME>
- <PATH>
- <COLOR> : (<FLOAT>, <FLOAT>, <FLOAT>, <FLOAT>)
- <2D-POSITION> : (<FLOAT>, <FLOAT>)
- <3D-POSITION> : (<FLOAT>, <FLOAT>, <FLOAT>)
- <POSITION-VECTOR> : 2차원 또는 3차원 점의 리스트(?)
- <2D-BOUNDING_BOX> : (<FLOAT>, <FLOAT>, <FLOAT>, <FLOAT>)
- <PROJ_DEFINITION>

2.1.4. referenced file types

- <NETWORK_FILE> : SUMO network file (netgenerate 또는 netconvert로 생성된)
- <ROUTES_FILE> : SMO routes file (duarouter, jtrrouter, 또는 손으로 생성된)
- <TYPE_FILE> : SUMO edge type file
- <OSM_FILE> : OpenStreetMap file

2.2. basic computer skills

SUMO 파일 다운로드한 bin 디렉토리에서 `start-command-line.bat`를 찾아서 실행해라. ros2 setup.bash와 유사한 스크립트로 보임.

환경변수 `SUMO_HOME`를 **bin**과 **tools**를 포함한 디렉토리 경로로 설정해주어야 한다.
`export`로 임시 변경하던지, **.bashrc** 설정으로 영구적으로 환경변수를 설정해주도록 하자.

```
netconvert --node-files=hello.nod.xml --edge-files=hello.edg.xml --output-
file=hello.net.xml
```

2.2.1. cli

SUMO 앱들은 plain executables이다. 그래서 터미널에서 명령어로 실행시킬 수 있다. 대부분 arg 인자 길이가 길어지므로, 아예 xml파일을 configuration 파일로서 사용할 수 있다.

"이름.sumocfg" 형식으로 다음과 같이 작성해주자.

```
<configuration>
    <input>
        <net-file value="test.net.xml"/>
        <route-files value="test.rou.xml"/>
        <additional-files value="test.add.xml"/>
    </input>
</configuration>
```

심지어 **--save-template**, **--save-configuration**, **--save-schema**를 통해 템플릿, 구성, 스키마 파일로서 저장할 수 있다.

xml 파일 내에서 `<net-file value="${NETFILENAME}.net.xml"/>` 이와 같이 환경변수 값을 포함시키도록 할 수 있다.

2.2.1.1. cli configuration options

- `--configuration-file <FILE>`
- `--save-configuration <FILE>`
- `--save-configuration.relative <BOOL>`
- `--save-template <FILE>`
- `--save-schema <FILE>`
- `--save-commented <BOOL>` : 템플릿, 구성, 스키마 파일에 주석을 저장할 것인가?

2.2.1.2. cli reporting options

- `--verbose <BOOL>`
- `--print-options <BOOL>`
- `--help <BOOL>`
- `--version <BOOL>`
- `--xlm-valdiation <STRING>` : "never", "auto", "always"로 xml 입력에 대한 스키마 검사 수행 여부를 설정한다.
- `--xlm-valdiation.net <STRING>` : 위와 비슷하나, SUMO network inputs에 대하여
- `--no-warnings <BOOL>`
- `--log <FILE>`
- `--message-log <FILE>`
- `--error-log <FILE>`
- `--language <STRING>`

2.2.1.3. cli random number options

- `--seed <INT>`
- `--random` : SUMO가 시드를 고르도록 만든다. 이것은 가능하다면 /dev/urandom 또는 current system time을 기준으로 시드를 얻을 것이다. 이것은 `--seed` 옵션보다 우선순위가 높다.

2.2.1.4. generating and reading edge-files

대부분 SUMO 패키지와 도구들이 읽고 생성하는 파일 형식은 xml이다. 이들 중 일부에 대해서는 xsd (XML schema definition) 스키마 형식이 존재한다.

2.2.1.5. writing files

파일이름을 다음과 같은 특수 노테이션으로 명시하면, 그러한 특수 파일 형태에다가 데이터를 써준다.

- "NUL" 또는 "/dev/null"로 쓰면 출력이 사실상 나오지 않는다.
- "<HOST>:<PORT>" 형식은 소켓 프로토콜을 통해 데이터를 전송한다.
- "stdout"이나 "-" 형식은 그저 cli 표준출력으로 출력한다.
- "stderr" 형식은 cli 표준출력에러로 출력한다.
- The special string 'TIME' within a filename will be replaced with the application start time

출력과 다르게, 현재 입력으로 socket과 stdin은 사용할 수 없다.

3. network building

3.1. SUMO Road Networks 소개

.net.xml 형식이며, 지도를 표현하고, net_file.xsd에 스키마가 정의되어 있다.

일반적으로 이 형식은 DAGs이다. nodes = junctions or intersections, edges = roads or streets.

- every street (edge) as a collection of lanes, including the position, shape and speed limit of every lane,
- traffic light logics referenced by junctions,
- junctions, including their right of way regulation,
- connections between lanes at junctions (nodes).

또한 옵션에 따라 districts, roundabout descriptions이 포함될 수 있다.

3.1.1. coordinates and alignment

3.1.2. edges and lanes

3.1.2.1. normal edges

```
<edge id="<ID>" from="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>"
function="<FUNCTION>">
    ... one or more lanes ...
</edge>
```

<FUNCTION>:

- "normal" : 일반 도로. 고속도나 거리에서의 도로.
- "connector": 현실에 존재하지 않는, 시뮬레이션 상 구분 요소. 사실상 normal과 다를 바 없다. macroscopic connector이다.
- "internal": intersection의 부분이다.
- "crossing"
- "walkingarea"

3.1.2.2. lanes

```
<lane id="<ID>_0" index="0" speed="<SPEED>" length="<LENGTH>" shape="0.00,495.05
248.50,495.05"/>
```

- index : 몇 번째 차로인지. rightmost부터 0 매기는 걸로 보임.
- speed : 최대 속도
- length : 길이
- shape : position vector(2차원 또는 3차원). 중앙선의 점 표현. 2개 이상 점 주어야 한다.

현재 엣지의 모든 차선은 같은 길이를 가진다고 함. netconvert가 이걸 명시적으로 덮어쓸 수 있게 해준다고 함.

3.1.2.3. internal edges

SUMO's Road Intersection Model https://elib.dlr.de/93669/1/LNCS_SUMOIntersections.pdf 참고

intersection(교차로) 안에 internal edge라는 가상의 좌회전/직진/우회전 등 라인이 있다고 가정한다. 교차로는 node(junction)으로만 단순히 표현하기에는 시뮬레이션 상 어려움이 있었다. 그래서 교차로에 인접하는 차선들을 a_0, a_1, a_2, .., b_0, b_1, b_2, .., 와 같이 라벨링 하고 이 때 -c_0와 같이 음수 표현으로 outgoing lane을 표현한다. 그리고 차량이 가질 수 있는 궤적을 internal edge로 표현하는 것이다.

- 이전에는 교차로 내부의 차량 궤적, 행동이 비현실적이였지만 이제 현실적인 동역학을 구현할 수 있다
- 향상된 통행권(right-of-way) 판단 로직
  - 여러 차량들이 교차로에서 상호작용하는 걸 도착 예상시간과 속도로 더 분석적인 방법으로 미리 알 수 있게 된다
  - 상충하는 경로(상대 차량)와의 시간 차이(안전 갭)를 계산하여, 차량이 안전하게 진입할 수 있는지 미래를 예측하여 판단합니다. 이로 인해 시간 단계에 독립적인 현실적인 접근 속도를 구현
- 교차로 내 차량 간 상호작용 (link leader)
  - 교차로 내부에서 경로가 겹치는 차량들 간 안전 거리 유지 위해
  - 기존의 차량 추종(car-following) 모델 재활용 -> 교착상태 및 충돌 방지
- 다양한 교차로 유형 및 운동자 행동 모델링
  - 신호등, 정지 신호, 양보 신호, 전방 정지(all-ay stop) 등 다양한 유형 지원
  - 인내심 매커니즘 : 오래 기다린 차량이 점점 더 공격적으로 행동하게 되는 현실적인 운전자 행태 구현
  - 교차로 진입 금지 규칙 구현 : 차량이 출구가 막힌 교차로에 진입하여(꼬리물기) 전체를 마비시키는 상황 방지

물론 `--no-internal-links` 옵션으로 네트워크에 포함되지 않도록 설정할 수도 있다.

internal edge의 ID가 합성되는 방식이 있다. `:<NODE_ID>_<EDGE_INDEX>`가 기본으로, 엣지 인덱스는 노드 주변 사방위 기준으로 번호가 매겨진다. 만약 진입엣지와 진출엣지가 여러개라면 `EDGE_INDEX + LANE_INDEX = CONNECTION_INDEX`와 같이 표현된다.

동일한 진출엣지, 진입엣지 쌍이더라도, 여러 연결이 있다면 하나의 internal edge 안에 여러 차선을 갖는 것과 비슷한 개념이 생겨난다. (dir = "s") 즉 직진으로 표시된 연결에서는 internal edge에서의 차선 변경이 허용된다!

좌회전시 진입차량 양보, 우회전시 보행자 양보하면서 기다리는 차선의 경우 internal junctions 같은 특수 케이스로 처리해준다. (우선권을 가진 다른 교통류를 기다리기 위해 대기하는 지점)
- 이것을 의도적으로 두 개의 internal edge로 분할했다.
- 첫째는, 대기 위치 전까지 주행하는 구간

- 둘째는, 대기 위치 후 교차로를 완전히 빠져나가기까지 주행하는 구간

3.1.2.4. stop offsets

https://en.wikipedia.org/wiki/Advanced_stop_line

advanced stop line을 찾아보아라. 이게 횡단보도도 페인트칠을 통해 너비가 늘어날 수 있다. 특히, 자전거 도로 페인트칠을 할 수도 있다. 어쨌든 기존 엣지에서 더 미리 정지해야하는 선이 있을 수 있다는 것이다. 그리하여 엣지던 차선이던 `stopOffset` 엘리먼트를 자식으로 포함할 수 있는 것이다. "bike box"와도 관련된다.

```
<stopOffset value="<distance in m.>" vClasses="<space-separated list of vClasses>" />
```

- vClasses: 이 정지오프셋 규칙이 적용되는 차량 클래스들 (블랙리스트)
- exceptions : 이것에 적용되지 않는 차량들 (화이트리스트)

3.1.3. traffic light programs

```
<tlLogic id="<ID>" type="<ALGORITHM_ID>" programID="<PROGRAM_ID>"
offset="<TIME_OFFSET>">
    <phase duration="<DURATION#1>" state="<STATE#1>"/>
    <phase duration="<DURATION#1>" state="<STATE#1>"/>
    ... further states ...
    <phase duration="<DURATION#n>" state="<STATE#n>"/>
</tlLogic>
```

자세한 것은 https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html
TLS-programs

3.1.4. junctions and right-of-way

3.1.4.1. plain junctions

정션은 다른 streams가 교차하는 영역을 표현한다. 그리고 통행권(right-of-way) 정책도 포함한다.

```
<junction id="<ID>" type="<JUNCTION_TYPE>" x="<X-POSITION>" y="<Y-POSITION>"
          incLanes="<INCOMING_LANES>" intLanes="<INTERNAL_LANES>"
          shape="<SHAPE>">
    ... requests ...
</junction>
```

- incLanes : [id list] 진입/진출 차선 ID 리스트
- intLanes : [id list] 교차로 내부 internal edge 내 차선 ID 리스트
- shape : [position list] 교차로 폴리곤 묘사
- customShape : [bool] 사용자가 커스터마이징 했는지 여부. True일 경우 netconvert나 netedit이 안 건들 것.

참고로 x, y 좌표는 정션의 중앙이 아닐 것이다. shape의 기준점이지 않을까 싶음.

3.1.4.2. requests

- edge : 도로의 한 구간
- lane : 엣지를 구선하는 차선
- connection : (교차로에서) 특정 진입 차선에서 특정 진출 차선으로 가기 위한 경로 또는 행동 (논리적 개념)

- link : connection이 실제로 지나는 교차로 내부의 물리적 공간 (구현체). 차량 개개는 connection으로 자기 경로를 상정하고, 그걸 link 상에서 다른 차량들을 고려해 교통을 파악하고자 한다.

```
  <request index="<INDEX>" response="<RELATIVE_MAJOR_LINKS>" foes="<FOE_LINKS>"
cont="<MAY_ENTER>"/>
```

- index : 현재 고려 중인 connection의 식별 id
- response : [bitset(string)] 상대적 주요 링크. 이 링크의 차량이 무조건 양보해야하는 우선순위가 높은 링크를 2진수 비트열로 표현 (애초에 정션마다 링크에 번호 매기는 규칙이 있을 것임)
- foes : [bitset(string)] 이 링크의 차량과 충돌 가능성이 있는 모든 링크를 나타냄. response의 슈퍼셋으로 볼 수도 있음. 물리적 위치 상 겹치면 다 표현한다고 보면 될 듯.
- cont : [bool] 현재 이 연결을 사용하는 차량이 존재하는가? (꼬리물기).

인덱스 정렬 순서
- edge 방향 : 북을 기준으로 시계 방향으로 edge 정렬
- lane 순서 : 하나의 edge에서는 right-most 차선부터 순서 매김
- connection 방향 : 가장 오른쪽 방향(우회전)부터 순서 매김

앞에 edge에서 설명한 것은, 말그대로 그 internal edge가 존재하고 어떠한지를 명시하는 거고, 여기서는 그 id를 갖다 써서 비트열 규칙 등을 매핑해주는 거라 보면 된다.

3.1.4.3. internal junctions

이것은 앞서 살펴본 plain junction + request라는 복잡한 매트릭스 통행권 규칙을 고려하지 않아도 된다. 적은 정보만 고려할 수 있는 장점이 있는 것이다.

```
<junction id="<ID>" type="internal" x="<X-POSITION>" y="<Y-POSITION>"
          incLanes="<INCOMING_PROHIBITING_LANES>"
          intLanes="<INTERNAL_PROHIBITING_LANES>"/>
```

- incLanes : [id list] internal junction이 위치한 교차로로 진입하는 차선들(internal 아닌 edge들)
  - 이 차선들에 차량이 접근하면 이 internal junction을 통과할 수 없다.
- intLanes : [id list] internal junction이 위치한 교차로 내부(internal edge)의 차선들
  - 역시나 이 차선들이 점유되면 내부 교차로를 통과할 수 없음

3.1.5. connections

3.1.5.1. plain connections

특정 진입차선으로부터 어느 진출차선으로 향할 수 있는지 그 연결 명시.

```
<connection from="<FROM_EDGE_ID>" to="<TO_EDGE_ID>" fromLane="<FROM_LANE_INDEX>"
toLane="<TO_LANE_INDEX>"
            via="<VIA_LANE_ID>" tl="<TRAFFIC_LIGHT_ID>" linkIndex="12" dir="r"
state="o"/>
```

- via : [lane id(string)] 이 논리적 커넥션이 대응되는 차선 id (internal edge의 차선)
- tl : [traffic light id(string)] 이 커넥션을 컨트롤하는 신호등
- linkIndex : [index (unsigned int)] 신호등 하 시그널 응답 채널? 인덱스
- dir
  - s : straight
  - t : turn
  - l : left
  - L : partially left

  - r : right
  - R : partially right
  - inavalid : no direction
- state (실시간성?)
  - “-” : dead end
  - “=” : equal
  - “m” : minor link
  - “M” : major link, traffic light only
  - “O” : controller off
  - “o” : yellow flashing
  - “Y” : yellow major link
  - “r” : red
  - “g” : green minor
  - “G” : green major

참고로 netconvert 없이 이런 커넥션을 만들 때 id 규칙을 잘 따라야 한다.
- ‘1_f_0’은 ‘1_v_0’을 통해 ‘1_t_0’으로 연결된다.
- ‘[from=1_f_0, to=1_t_0 via=1_v_0]’와 ‘[from=1_v_0 to=1_t_0]’이 둘 다 존재해야 한다? 후속 커넥션도 묘사해야 하는 것으로 보임.
- 또는 ‘[from=1_f_0 to=1_v_0]’ 같이 불필요한 커넥션을 추가해도 안됨.
- 이 까다로운 규칙을 지켜야 네트워크 로딩이 될 것으로 보임.

커넥션 인덱스 종류
- junction index (교차로 인덱스)
  - 수정 불가능
  - 시계 방향으로 인덱스 부여
  - requests 우선순위에 사용됨
- TLS index (신호등 인덱스)
  - 신호등에 의해 제어되는 커넥션에만 부여
  - 여러 교차로를 함께 제어하는 통합 신호등에서는 junction index와 다를 수 있음
  - 신호 그룹 생성에 사용 (같은 상태를 공유하는 여러 커넥션)

3.1.6. roundabouts

4. simulation

5. additional tools

6. theory

7. application manuals

8. appendices

8.1. file extensions

대부분 xml 파일들에 대하여 xsd 명세가 있으므로 https://sumo.dlr.de/docs/Other/File_Extensions.html 방문해서 살펴보도록 하자.

참고로 xsd 파일들은 <SUMO_HOME>/data/xsd 디렉토리에 있다.
https://sumo.dlr.de/xsd/ 이 링크에서 확인할 수도 있다.

8.1.1. native SUMO Files

- configuration files
  - *.sumocfg : **sumo**와 "sumo-gui"의 구성 파일
  - *.netecfg : "netedit"의 구성 파일
  - *.netccfg : "netconvert"의 구성 파일
  - *.netgcfg : "netgenerate"의 구성 파일
  - *.duarcfg : "duarouter"의 구성 파일
  - *.jtrrcfg : "jtrrouter"의 구성 파일
  - *.dfrocfg : "dfrouter"의 구성 파일
  - *.od2tcfg : "od2trips"의 구성 파일
  - *.acticfg : "activitygen"의 구성 파일
- data files
  - *.net.xml : network file
  - *.rou.xml : routes file
  - *.rou.alt.xml : route alternatives file
  - *.add.xml : additional files
    - 이에 대하여 신호등 전용 xsd도 있음
  - *.edg.xml : netconvert edges file
  - *.nod.xml : netconvert nodes file
  - *.con.xml : netconvert connection file
  - *.typ.xml : netconvert edge types file
  - *.trips.xml : trip definitions for duarouter, sumo
  - *.flows.xml : flow definitions for jtrrouter, duarouter, sumo
  - *.turns.xml : turn and sink definitions for jtrrouter
  - *.taz.xml : traffic analysis zones (or disstricts) file mainly for od2trips, duarouter, sumo
- output files
  - *.xml (inductive loop output)
  - *.xml (areal lane detector output)
  - *.xml (areal lane detector output)
  - *.xml (emissions output)
  - *.xml (fcd output)
  - *.xml (full output)
  - *.xml (meandata output)
  - *.xml (netstate output)
  - *.xml (queue output)
  - *.xml (summary output)
  - *.xml (tripinfo output)
  - *.xml (vtypeprobe output)
  - 참고로 실험적으로나마 csv, Parquet 형식의 tabular output을 배노내는 옵션도 존재하는 것으로 보인다 `--fcd-output` 옵션을 사용하면 csv나 parquet 형식으로 내보낼 수 있다. https://sumo.dlr.de/docs/TabularOutputs.html 에서 더 살펴보도록 하자.
- other files
  - *.xml (edge diff)

SUMO FCD 출력 파일에 대해 대략적으로 느낌을 보자. 다음은 .fcd.xml 파일의 구성 예시이다.

```
<fcd-export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/fcd_file.xsd">
    <timestep time="0.00"/>
    <timestep time="1.00"/>
    <timestep time="2.00"/>
...
    <timestep time="97.00"/>
    <timestep time="98.00"/>
    <timestep time="99.00"/>
    <timestep time="100.00">
        <vehicle id="always_left.0" x="5.10" y="498.35" angle="90.00"
type="DEFAULT_VEHTYPE" speed="0.00" pos="5.10" lane="1fi_0" slope="0.00"/>
        <vehicle id="always_right.0" x="501.65" y="5.10" angle="0.00"
type="DEFAULT_VEHTYPE" speed="0.00" pos="5.10" lane="3fi_0" slope="0.00"/>
        <vehicle id="horizontal.0" x="994.90" y="501.65" angle="270.00"
type="DEFAULT_VEHTYPE" speed="0.00" pos="5.10" lane="2fi_0" slope="0.00"/>
    </timestep>
...
```

결국 시뮬레이터라 함은, 타임스텝에 대해 계산을 해줄 뿐인 것이다. sumo executable은 시각화 요소가 없다. 이러한 계산을 하는 것이 시뮬레이터의 a-to-z 라고 할 수도 있겠다.

물론 parquet data type 중 표현하기 어려운 구조가 있다
- elements with optional attributes (i.e. edgeData defaults)
- different elements on the same level (i.e. elements `<walk>` and `<ride>` of a persons plan)

8.1.2. imported files
- *.osm : OpenStreetMap XML databases
- *.xodr : OpenDRIVE xml network files
- *.inp : VISSIM network files
- *.net : VISSIM network files
- *.shp, *.shx, *.dbf : ArcView-network descriptions (shapes, shape indices, definitions)
- *.xml (MATSim road networks)

8.1.3. exported files
- *.xml
  - MATSim road networks
  - OMNET mobility-traces
  - SHawn snapshot-files
- *.xodr
- *.tcl : ns2/ns3 trace-files, activity-files, and mobility-files (Tools/TraceExporter)
- *.dri, *.str, *.fzp, *.flt : PHEM input files (Tools/TraceExporter)
- unknown : GPSDAT (Tools/TraceExporter)

9. xsd 상세 설명

9.1. net_file.xsd

`<xsd:element name="net" type="netType">`로 구성된다.

- netType
- edgeType
- spreadTypeType
- laneType

- juncitonType
- requestType
- stopOffsetType
- connectionType
  - [attr] from: xsd:string
  - [attr] to: xsd:string
  - [attr] fromLane: xsd:string
  - [attr] toLane: xsd:string
  - [attr] pass: boolType
  - [attr] keepClear: boolType
  - [attr] centPos: floatType
  - [attr] visibility: floatType
- prohibitionType
  - [attr] prohibitor: xsd:string
  - [attr] prohibited: xsd:string
- roundaboutType
  - [attr] nodes: xsd:string
  - [attr] edges: xsd:string
- neighType
  - [attr] lane: xsd:string