VEHICULAR FLOW SIMULATORS BASED ON COMPLEX QUEUING SYSTEMS

LEONARDO PASINI AND SANDRO FELIZIANI

Department of Computer Science, University of Camerino, Via Madonna delle Carceri 9, 62032 Camerino (MC), Italy
leonardo.pasini@unicam.it

(Received 20 September 2013; revised manuscript received 15 October 2013)

Abstract: The crux of this paper is to demonstrate how it is possible to model systems of vehicular traffic by means of complex queuing network models. The objects comprising the library that we define in this work all follow the approach whereby they are composed of one or more queuing networks of service systems that are traversed by customers (vehicles).

Keywords: traffic systems, queueing networks, modeling and simulation

1. Introduction

In this paper we describe an object library for the construction of simulators of systems involving urban vehicular traffic flows. Such systems are formed by a network of intersections and roundabouts mutually connected by urban streets, and by a set of traffic flows that cross them.

The architecture of the main objects of this library is described by means of a queuing network model. The flow of a vehicle in the road network is simulated by a sequence of events that describe its trajectory in a queueing network of service systems.

This work was envisioned in our previous paper [1], where we demonstrated a graphical application for the construction of files describing street intersections. Such files are formatted as a list of instances of library objects which constitute the components of the intersection. In this work we define library objects on which the previous work was based and subsequently extend the library to allow it to describe urban street systems that are more complex compared to those dealt with in our previous publications [2, 3]. The novelty of this work lies in the definition of a method for the description of urban vehicular traffic flows at roundabouts and a more general description of traffic flow in systems with multiple intersections and roundabouts mutually connected by urban streets.
2. An urban traffic system

We shall consider a problem consisting in modeling and simulating a street system illustrated in Figure 1. The system in question is formed by four intersections, two of which are roundabouts, and a number of urban streets that are lined with major traffic flows.

In Figure 2 we show a schematic of a system in which the intersections, urban streets and the corresponding vehicular flows are identified.

Figure 2 illustrates how the vehicular flows comprising the system can be labeled by integer numbers and how they are denoted by colours according to the following convention:

- green signifies a vehicular flow entering a street system from the outside;
- red signifies a vehicular flow exiting a street system to the outside;
- yellow signifies a vehicular flow within a street system.

The street network shown in Figure 2 comprises four nodes that are four street intersections. Each intersection is labelled with a tag, with the following relationships in effect:

- Intersection 1 is a roundabout that regulates the following traffic flows:
  - receives input flows: 1, 3, 5, 9 originating outside the system, and flow 7, which is within the system;
  - routes outgoing flows: 2, 4, 8, 10 to the outside of the system, and flow 6, which is within the system.
Figure 2. Schematic of vehicular flows in the Siena Nord street system
• Intersection 2 is a non-signalised street intersection which regulates the following traffic flows:
  – receives input flows: 13 originating outside the system, and flows 12 and 19, which are within the system;
  – routes outgoing flows within the system: 14 and 20.

• Intersection 3 is a roundabout that regulates the following traffic flows:
  – receives input flows: 17, 18 originating outside the system, and flow 14, which is within the system;
  – routes outgoing flows: 15 and 16 to the outside of the system, and flows 7 and 19, which are within the system.

• Intersection 4 is a non-signalised street intersection which regulates the following traffic flows:
  – receives flows 11 and 20, which are within the system;
  – routes outgoing flow 21 to the outside of the system.

Each traffic flow in Figure 2 identifies a direction for vehicles traveling on a street in the network. For example Fiume Street (“Strada Fiume”) in Figure 2 comprises two directions of travel identified by flows 9 and 8. Such flows are represented in our simulation model by a specific object: Road, which serves to describe certain physical characteristics of a carriageway, e.g. its length, width, etc. This in turn makes it possible to realistically describe the dynamics of vehicular flows on the streets of the system.

3. Specifying intersections

In this paragraph we illustrate the procedure we use for implementing models for specifying intersections in the system under study.

We separately study single intersections, each of which can be seen as an independent object composed of certain internal elements (sections), and connected to other intersections by means of streets. We identify the inputs and outputs of each intersection and subsequently trace the possible trajectories that vehicles can follow during the crossing.

For the purpose of specifying an intersection we will define the following types of sections:

• **input sections** are denoted with green and constitute areas of entry into an intersection; these correspond to lanes that serve to funnel the traffic near an entrance to an intersection;

• **output sections** are denoted with red; these delineate exits from the intersections and connect the intersection with the streets that exit it;

• **inner sections** are denoted with yellow (in the case of normal intersections) or represented by arrows (in the case of roundabouts); these are understood as road surfaces internal to the intersection over which the vehicles pass during the crossing.
The following images show, for each intersection of the system illustrated in Figures 1 and 2, the input and output sections, the possible trajectories that a vehicle can take during the crossing, as well as the inner sections that are traversed for every path through the intersection. With every section a numerical id is associated, which identifies the section in the file that describes the intersection.

3.1. Roundabout 1

Figure 3. Input and output sections of roundabout 1

Figure 3 shows that roundabout 1 has 5 input sections and 5 output sections. Below we characterise the input sections (cf. Figures 2 and 3):

- Input section 1: entry point into the roundabout for incoming vehicles from Chiantigiana state highway (from the direction of Castellina in Chianti);
- Input section 2: entry point into the roundabout for incoming vehicles from Via Val d’Aosta;
- Input section 3: entry point into the roundabout for incoming vehicles from Via Montecelso;
- Input section 4: entry point into the roundabout for incoming vehicles from Chiantigiana state highway (from the direction of Stellino);
- Input section 5: entry point into the roundabout for incoming vehicles from Strada Fiume.

The output sections can be characterised as follows (cf. Figures 2 and 3):

- Output section 1: exit from the roundabout leading to Via Val d’Aosta;
- Output section 2: exit from the roundabout leading to Via Montecelso;
- Output section 3: exit from the roundabout leading to Chiantigiana state highway (in the direction of Stellino);
- Output section 4: exit from the roundabout leading to Strada Fiume;
- Output section 5: exit from the roundabout leading to Chiantigiana state highway (in the direction of Castellina in Chianti).
Figure 4. Inner sections of roundabout 1

Figure 4 shows roundabout 1 with all the input and output sections, as well as with all the inner sections delineated.

3.2. Roundabout 2

Figure 5. Input and output sections of roundabout 2

Figure 5 shows that roundabout 2 has 4 input sections and 4 output sections. Below we analyse the input sections (cf. Figures 2 and 5):

- Input section 1: entry point into the roundabout for incoming vehicles from a branch of Via Cassia Nord;
- Input section 2: entry point into the roundabout for incoming vehicles from Via Delle Province;
- Input section 3: entry point into the roundabout for incoming vehicles from Via Fiorentina (inner lane);
- Input section 4: entry point into the roundabout for incoming vehicles from Via Fiorentina (outer lane).
The output sections can be characterised as follows (cf. Figure 2 and 5):

- Output section 1: exit from the roundabout leading to Via Delle Province;
- Output section 2: exit from the roundabout leading to Via Fiorentina;
- Output section 3: exit from the roundabout leading to a branch of Via Cassia Nord which continues through intersection 2;
- Output section 4: exit from the roundabout leading to Chantigiana state highway (in the direction of Stellino) which continues through roundabout 1.

Figure 6 shows roundabout 2 with all the input and output sections, as well as with all the inner sections delineated.

![Figure 6. Inner sections of roundabout 2](image)

### 3.3. Intersection 2

Figure 7 shows that intersection 2 has 3 input sections and 2 output sections. Referring to Figure 2, we analyse the input sections below:

![Figure 7. Input and output sections of intersection 2](image)
• Input section 1: entry point into the intersection for incoming vehicles from Chiantigiana state highway (from the direction of Stellino);
• Input section 2: entry point into the intersection for incoming vehicles from Via Cassia Nord heading south through roundabout 2;
• Input section 3: entry point into the intersection for vehicles leaving roundabout 2 heading onto Via Cassia Nord.

Referring again to Figure 2, we analyse the output sections of intersection 2:
• Output section 1: exit leading to input section 2 of intersection 4 in the direction of Via Cassia Nord;
• Output section 2: exit leading to input section 1 of roundabout 2.

In this context, a path through an intersection can be described as an ordered sequence of sections of that intersection. The sequence begins with an input section and finishes with an output section. The intermediate sections in the sequence are the inner sections of the intersection that a vehicle must cross while moving from an input section to the output section. Figures 8 and 9 illustrate a path through intersection 2 and a sequence of sections that comprise it. In this specific case, a vehicle entering through input section 1 can only proceed forward and exit through output section 2.

Figures 10 and 11 illustrate the only possible path for crossing the intersection for a vehicle entering it through input section 2.

Finally, Figures 12 and 13 illustrate the only possible path for crossing intersection 2 for a vehicle entering it through input section 3.

3.4. Intersection 4

Figure 14 shows that intersection 4 has 2 input sections and only one output section. Below we describe the input and output sections referring the general schematic of the system shown in Figure 2:
• Input section 1: entry point into the intersection for incoming vehicles from Chantigiana state highway (from the direction of Stellino) heading to Via Cassia Nord;
• Input section 2: entry point into the intersection for incoming vehicles from roundabout 2 heading to Via Cassia Nord;
• Output section 1: exit leading to Via Cassia Nord.

There are only two possible paths through this intersection – these are illustrated in Figures 15–16 and Figures 17–18, respectively.

**Figure 15.** First path for crossing the intersection

**Figure 16.** Inner sections of first path

**Figure 17.** Second path for crossing the intersection

**Figure 18.** Inner sections of second path

4. Library objects

The library objects are defined using the programming language QNAP2 [4]. In this paragraph, for each of these, we show the corresponding definitions and describe their internal variables. We note that all objects belonging to the first group described below, up until the INTERSECTION object, have the following property. Each object of this type is globally identified, through an internal variable, in the model of the system. On the other hand, all objects that follow
INTERSECTION are uniquely identified locally in the scope of the intersection to which they belong.

The VEHICLE object is defined as a subtype of the CUSTOMER object, which is a predefined object type in QNAP2 used for representing a generic user of a queueing system. The VEHICLE object inherits the properties of CUSTOMER and describes a generic vehicle that moves in the street traffic system. The code defining this object is:

```plaintext
CUSTOMER OBJECT VEHICLE(ID);
  INTEGER I, ID_PAT, ID_INT, MP_SC, ID;
  REAL SPEED;
END;
```

Below we describe its internal variables. As the vehicle traverses a trajectory in the system over the course of a simulation, the values of these internal variables change accordingly:

- **I**: during the crossing of an intersection, indicates the number of sections already crossed;
- **ID_PAT**: identifies the path for crossing an intersection taken by the vehicle;
- **ID_INT**: identifies the intersection which the vehicle enters;
- **MP_SC**: identifies the next Multiplexer to which the vehicle will be routed;
- **ID**: identifies the vehicle;
- **SPEED**: the vehicle’s velocity.

The FLOW object generates vehicular flow for an input of the system. In our models this object is used to simulate the arrival of vehicles which enter the system from a predetermined direction. The average time between arrivals of two consecutive vehicles is calculated based on the measurements of vehicular flows provided by Municipalities.

The code defining the FLOW object is:

```plaintext
OBJECT FLOW(ID,TR);
  QUEUE F_SOURCE, ROUTING;
  INTEGER ID, ID_ROAD;
  REAL TR;
  REF QUEUE ENTRY;
  REF CUSTOMER NEW_VE;
END;
```

Below we describe the properties of a FLOW object:

- **ID**: unique identifier of the object;
- **F_SOURCE**: the queue generating the vehicles;
- **ROUTING**: the queue routing the vehicles into the system;
- **ID_ROAD**: identifier of the street that acts as an entry point to the system;
- **TR**: average time between arrivals of two consecutive vehicles;
- **ENTRY**: a pointer to the entry point to the system;
- **NEW_VE**: a pointer to a VEHICLE object.
The ROAD object identifies a one-way carriageway. The code defining it is:

```
OBJECT ROAD(ID, EXT, LENG_RO);
QUEUE STRETCHR, QUECONTR;
REAL TIME_RUN, TMP, TMP2;
REAL LENG_RO, VEL_REAL;
INTEGER ID, EXT, INTERSEC, ID_MP, ID_MP_RO, MAX_VEIC, R_NB_IN,
     R_NB_OUT;
FLAG SAT;
STRING NAME;
END;
```

The characteristics of the object are described below:

- **ID**: identifier of the object;
- **STRETCHR**: the service queue of the vehicles;
- **QUECONTR**: the service queue of the control signal;
- **TIME_RUN**: time spent on a route, calculated during the simulation;
- **TMP**: temporary variable for calculating the time spent on a route;
- **TMP2**: temporary variable for converting **MAX_VEIC** from floating-point to integer;
- **LENG_RO**: the length of the route, expressed in km;
- **VEL_REAL**: actual velocity of the vehicle on a route, calculated during the simulation;
- **EXT**: a boolean variable specifying whether a street is an output channel of the system;
- **INTERSEC**: a boolean variable specifying whether a street terminates in an intersection Multiplexer or in a road Multiplexer;
- **ID_MP**: identifies the intersection multiplexer into which the street enters;
- **ID_MPRO**: identifies the road Multiplexer into which the street enters;
- **MAX_VEIC**: the maximum number of vehicles that the road can hold;
- **R_NB_IN**: the number of vehicles entering, according to the data obtained from the measurements performed by the Municipality;
- **R_NB_OUT**: the number of vehicles exiting, according to the data obtained from the measurements performed by the Municipality;
- **SAT**: a boolean variable which indicates, during the simulation, if a road is saturated;
- **NAME**: the name of the road.

The MP_INT object defines an intersection multiplexer. In our model this object describes the transition phase of vehicles which at the end of a journey through a road are preparing to choose a lane for entering an intersection. In particular, the algorithm of the servicing unit internal to this object routes the vehicle in transit into the corresponding input section of an intersection.

The corresponding definition is:

```
OBJECT MP_INT (ID, NB_INP);
```
The properties of the multiplexer are the following:

- **ID**: identifier of the Multiplexer;
- **MP_RQ**: the transit request queue of the Multiplexer;
- **MP_VQ**: the vehicle queue of the Multiplexer;
- **NB_INP**: the number of input sections attached to the Multiplexer;
- **NCI(NB_INP)**: a vector containing the IDs of the accessible input sections;
- **ID_INTER**: intersection identifier;
- **PRI(NB_INP)**: a vector of routing probabilities for input sections.

The **MP_ROAD** object defines a road multiplexer. This object routes vehicles at the end of a road which branches into a number of distinct streets. A servicing unit internal to this object assigns a direction to a vehicle, thereby routing it to one of the possible roads.

The code defining the **MP_ROAD** object is shown below:

```plaintext
OBJECT MP_ROAD(ID, NB_R_OUT);
 QUEUE MP_RO_VQ, MP_RO_RQ;
 INTEGER ID, NB_R_OUT, ID_RO(NB_R_OUT);
 REAL PRI(NB_R_OUT);
END;
```

The properties of the **MP_ROAD** object are listed below:

- **ID**: an identifier of a road Multiplexer;
- **MP_RO_VQ**: the vehicle queue of the Multiplexer;
- **MP_RO_RQ**: the transit request queue of the Multiplexer;
- **NB_R_OUT**: the number of exit roads;
- **ID_RO(NB_R_OUT)**: a vector of identifiers of exit roads;
- **PRI(NB_R_OUT)**: a vector of probabilities of routings into subsequent roads.

The **INTERSECTION** object represents a crossing or a roundabout. In our simulation model, the structure of an object of this type is carried out starting from instances of a number of object types that will be described further in the text. These objects constitute components of an intersection and are uniquely identified in the scope of the intersection that encompasses them.

The code defining the **INTERSECTION** object is shown below:

```plaintext
OBJECT INTERSECTION(ID);
 INTEGER ID, NB_SEM, NB_INPS, NB_SEC, NB_OUT, NB_PAT, D_SEM, ROUND;
END;
```

Below we describe the variables internal to this object:

- **ID**: a unique identifier of the intersection in the simulation model;
- **NB_SEM**: the number of traffic lights (semaphores) in the intersection;
• \text{NB_INPS}: the number of input sections in the intersection;
• \text{NB_SEC}: the number of inner sections in the intersection;
• \text{NB_OUT}: the number of output sections in the intersection;
• \text{NB_PAT}: the number of paths for crossing the intersection;
• \text{D_SEM}: the total duration of one traffic light cycle in the intersection;
• \text{ROUND}: a boolean variable that indicates if the intersection is a crossroad or a roundabout.

The \text{SEMCOMP} object identifies a semaphore (traffic lights). If a street intersection is signalised, a single \text{SEMCOMP} object identifies a semaphore that controls the access of vehicles into the intersection from a single entry lane.

The code defining the \text{SEMCOMP} object is shown below:

\begin{verbatim}
OBJECT SEMCOMP(ID);
  INTEGER ID, FR1, FR2, FG1, FG2;
  QUEUE R1, R2, G1, G2, Y;
  REF FLAG RF(ID);
  REAL TR1, TR2, TG1, TG2, TOT, D;
  REF SEMCOMP RSC;
END;
\end{verbatim}

Below are its properties:
• \text{ID}: identifier of a semaphore in the intersection;
• \text{FR1}, \text{FR2}, \text{FG1}, \text{FG2}: boolean variables needed for initialising the semaphore;
• \text{R1}, \text{R2}, \text{G1}, \text{G2}, \text{Y}: the queues traversed by a signal that reproduces the traffic light cycle;
• \text{RF}: a reference to the input section controlled by the semaphore;
• \text{TR1}, \text{TR2}, \text{TG1}, \text{TG2}: the times when the traffic light shows, respectively, red and green;
• \text{TOT}: the sum of durations for red and green;
• \text{D}: indicates the total duration of the traffic light cycle;
• \text{RSC}: is a reference to the \text{SEMCOMP} object.

The \text{INPSEC} object represents a single entry lane into an intersection, which we term an entry section. A street intersection can belong to one of the following classes: simple intersection, signalised intersection, roundabout. In each of these cases, an \text{INPSEC} object identifies one of the entry lanes into an intersection which a vehicle needs to traverse in order to initiate crossing an intersection.

The code defining the \text{INPSEC} object is shown below:

\begin{verbatim}
OBJECT INPSEC(ID, NDIRECT, SERV);
  QUEUE INP_VQ, INP_RQ;
  INTEGER NDIRECT, ID, NCP(NDIRECT), SEM_FLAG, NB_VE;
  REAL SERV, PR(NDIRECT);
  FLAG SEMAPH;
  REF QUEUE REQRT(NB_PAT);
  REF QUEUE VHERT(NB_PAT);
\end{verbatim}
INTEGER PRIORITY(NB_PAT);
END;

The main properties of this object are:

- **ID**: a unique identifier of the object in the intersection;
- **INP_VQ**: a queue of vehicles in the input section;
- **INP_RQ**: a queue of transit request in the input section;
- **NDIRECT**: number of possible directions along which a vehicle can be routed through the input section;
- **NCP(NDIRECT)**: a vector containing possible crossing paths that start from the input section;
- **PR(NDIRECT)**: a vector of probabilities of routing through the above paths;
- **VE**: maximum number of vehicles that the input section can hold;
- **SEM_FLAG**: a boolean variable indicating if the traffic in the input section is controlled by a semaphore;
- **SERV**: the time it takes for a vehicle to cross the input section;
- **SEMAPH**: a variable for connecting to the semaphore controlling the input section;
- **REQRT(NB_PAT)**: a pointer to the request queue of the first inner section of the path for crossing the intersection identified by \(NB\_PAT\);
- **VHERT(NB_PAT)**: a pointer to the vehicle queue of the first inner section of the path for crossing the intersection identified by \(NB\_PAT\);
- **PRIORITY(NB_PAT)**: the priority level with which the vehicle enters the intersection.

The **SECTION** object represents an internal section of an intersection. It represents the section of the trajectory taken by vehicles crossing an intersection between an input section and an output section.

The code defining this object is shown below:

```c
OBJECT SECTION(ID);
    INTEGER ID;
    QUEUE VQ, RQ;
    REAL SERV;
    REF QUEUE REQRT(NB_PAT);
    REF QUEUE VHERT(NB_PAT);
    INTEGER PRIORITY(NB_PAT);
    INTEGER NB_V;
    REF OUTSEC EXIT;
END;
```

The main characteristics of this object are described below:

- **ID**: a unique internal identifier within an intersection object;
- **VQ**: the vehicle queue that simulates the act of crossing the section;
- **RQ**: the queue holding requests for transit through the section issued by vehicles;
- **SERV**: the time it takes to traverse the section, depending on the length of the inner section;
L. Pasini and S. Feliziani

- **REQRT(NB_PAT)**: a pointer to the request queue of the subsequent SECTION object for the path identified by NB_PAT;
- **VHERT(NB_PAT)**: a pointer to the vehicle queue of the subsequent SECTION object for the path identified by NB_PAT;
- **PRIORITY(NB_PAT)**: the priority level which a vehicle acquires when entering a section following the path identified by NB_PAT;
- **NB_V**: the maximum number of vehicles which the inner section can hold, its value depends on the length of the section;
- **EXIT**: a pointer to an output section.

The **OUTSEC** object represents an outer section of an intersection. This object represents a single lane that a vehicle must take in order to exit the intersection at the end of its crossing path and be routed to an external road.

The code defining this object is shown below:

```plaintext
OBJECT OUTSEC(ID);
    INTEGER ID, ID_ROAD;
    QUEUE VQ, RQ;
    REAL SERV;
    REF QUEUE EXIT;
END;
```

The properties of this object are listed here in detail:

- **ID**: an internal variable which identifies the exit section within an intersection object;
- **ID_ROAD**: identifies the road into which the exit section routes vehicles;
- **VQ**: a vehicle queue for vehicles within the section;
- **RQ**: a queue holding transit requests for the section;
- **SERV**: the time it takes to traverse the section;
- **EXIT**: a pointer to a road for exiting the section.

The **PATH** object represents a path for crossing an intersection that can be taken by a vehicle entering from a given input section and leaving the intersection through an exit. A **PATH** object identifies an ordered sequence of inner sections that a vehicle takes in its trajectory through an intersection, from its entry point to its exit.

The code defining a **PATH** object is shown below:

```plaintext
OBJECT PATH(ID, L);
    INTEGER L, ID, I;
    REF SECTION DESTI(L);
    INTEGER PRI0(L);
    REF INPSEC INP_SEC;
    REF OUTSEC OUT_SEC;
END;
```

The main properties of this object are:

- **ID**: a unique identifier of the path in the intersection;
• \( L \): the length of the path, that is the number of sections it comprises;
• \( \text{DESTI}(L) \): contains the pointers to successive inner sections of the path that need to be traversed;
• \( \text{PRIO}(L) \): is the priority level that a vehicle acquires when entering subsequent sections in the path;
• \( \text{INP\_SEC} \): is a pointer to the input section of the path;
• \( \text{OUT\_SEC} \): is a pointer to the output section of the path.

5. Queues internal to objects

Certain library objects defined in the previous paragraph contain internal variables of the queue type. Such variables can function as servicing units. Objects which have this property can be traversed by clients, who, upon entering a servicing unit, activate corresponding service algorithms. A service station based on a variable that is an internal queue of a library object is implemented using the station template mechanism, which is a programming idiom of QNAP2.

The above mechanism gives means to define, in a general fashion, the structure of queue-based servicing units that are contained in library objects. A library object and its internal queue variable can be used to define, by means of a station template, the structure of a servicing unit associated with the internal variable. This is done in a way that is consistent across all instances of this object in the simulation model.

In this paragraph we describe the station templates that we defined for queue variables contained in library objects. The description is structured according to the type of library objects containing internal queue variables. For each object type listed below the corresponding QNAP2 code for defining the station template associated with the internal queue variables will be given. Moreover, in subsequent paragraphs the code for defining each station template will be followed by a description of the service station.

5.1. The FLOW object

This object contains two internal queue variables: \( \text{F\_SOURCE} \) and \( \text{ROUTING} \). Below we report the code for defining the station templates and their corresponding descriptions.

```
/STATION/
NAME=*FLOW.F\_SOURCE;
TYPE=SOURCE;
SERVICE= BEGIN
  CST(TR);
  NEW_VE := NEW(Vehicle);
  WITH NEW_VE::Vehicle DO BEGIN
    SPEED := V\_SP;
    TRANSIT(NEW_VE, FLOW.ROUTING, VHE);
  END;
```

The F_SOURCE station is of the SOURCE type. This type of station generates, based on the station service algorithm described by the SERVICE parameter, a client of the CUSTOMER type every TR units of time. Each CUSTOMER generated by the source, in turn, generates a VEHICLE object to which a standard velocity V_SP is assigned. The VEHICLE object is then sent to the ROUTING queue of the same object, whose task is to route vehicles into the next road that is an entry point to the network of street intersections.

The ROUTING queue is structured as a single server with a First-In First-Out policy. Each vehicle passing through this unit checks the value of the SAT variable of the street into which the vehicles generated by the Flow object are routed. The WAIT instruction in the service algorithm of this queue blocks the routing of the vehicle in service, if the destination road is saturated; that is, if the control variable is in the UNSET state, the instruction blocks until the variable returns to the SET state. The TRANSIT instruction transfers the vehicle to the STRETCHR queue of the ROAD object, which is connected to the flow by means of the ENTRY pointer.

5.2. The ROAD object

This object represents a channel of vehicular flow that runs in one direction of travel along a road in the real system. We will term a ROAD object “external” when the vehicular flow that runs along it exits the system. In this case the internal variable EXT of the object will have the value of 1.
The QUECONTR queue is an internal variable of the ROAD object. Each of its instances is initialised in such a way that it contains a client of the CONTROLLER type. It serves to control the state of the street every 3 seconds. If the number of vehicles it contains is below the maximum capacity of the street, the value of the SAT variable is set to open, otherwise it evaluates to closed.

/STATION/
NAME=*ROAD.STRETCHR;
TYPE= SERVER, MULTIPLE(LENG_RO / 0.0045);
SCHED=FIFO;
SERVICE = BEGIN
  VEL_REAL := CUSTOMER::Vehicle.SPEED;
  TIME_RUN := (LENG_RO / VEL_REAL) * 3600;
  CST(TIME_RUN);
  IF (EXT = 1) THEN
    TRANSIT(OUT);
  IF (INTERSEC=1) THEN
    BEGIN
      P(MP#(ID_MP).MP_RQ);
      TRANSIT(MP#(ID_MP).MP_VQ);
    END
  ELSE
    BEGIN
      P(MP_R#(ID_MP_RO).MP_RO_RQ);
      TRANSIT(MP_R#(ID_MP_RO).MP_RO_VQ);
    END;
  END;

The STRETCHR queue is a channel along which vehicles flow, it contains a multiple server. The number of servers it contains is equal to the maximum number of vehicles the street can hold. The service algorithm calculates the time it takes to traverse along the length of the street. When this time elapses, the service algorithm executes the following checks:

- If the street is external, the vehicle exits the system;
- If the street terminates with an intersection Multiplexer, the current vehicle issues a request (P) to the resource queue (MP_RQ) of the corresponding intersection multiplexer and, having obtained the resource, is routed into the vehicle queue of the multiplexer (MP_VQ);
- If the streets terminates with a road Multiplexer, the current vehicle issues a request to the resource queue of the corresponding road multiplexer (MP_RO_RQ) and, having obtained the resource, is routed into the vehicle queue of the multiplexer (MP_RO_VQ).
5.3. The MP_INT object

This object is a unit used in the simulation model for connecting a Road object with an Intersection object. It has two internal queue variables: MP_VQ and MP_RQ. The code for defining the station templates is given below.

```
/STATION/
NAME=*MP_INT.MP_VQ;
TYPE=SERVER, MULTIPLE(2);
SERVICE= BEGIN
    CST(2.0);
    CUSTOMER::Vehicle.MP_SC:= DISCRETE(NCI(1 STEP 1 UNTIL NB_INP),
        PRI(1 STEP 1 UNTIL NB_INP));
    CUSTOMER::Vehicle.ID_INT:=ID_INTER;
    P(INP#(ID_INTER,CUSTOMER::Vehicle.MP_SC).INP_RQ);
    V(INCLUDING(QUEUE)::MP_INT.MP_RQ);
    TRANSIT(INP#(ID_INTER,CUSTOMER::Vehicle.MP_SC).INP_VQ);
END;
```

The station defines a multiple server with 2 servers. After a constant service time, the service algorithm assigns to the vehicle an input section selected from a list of accessible input sections according to a known probability distribution. The service algorithm then issues an access request to the resource queue of the input section assigned to the vehicle. Once the resource is obtained, it is released from the multiplexer and the vehicle is sent to the INP_VQ queue of the input section of the intersection.

```
/STATION/
NAME=*MP_INT.MP_RQ;
TYPE=RESOURCE,MULTIPLE(2);
```

MP_RQ is a resource queue which regulates the accesses to the intersection multiplexer. It serves to handle the transit requests of vehicles, limiting the access to the multiplexer to no more than two vehicles at a time.

5.4. The MP_ROAD object

This object is used in the construction of the simulation model when it is necessary to connect distinct Road objects. As an example consider a case when a street branches into two streets, as is the case with street 6 shown in Figure 2. The internal queue variables of this object are MP_RO_VQ and MP_RO_RP. The code for defining the station templates corresponding to these two queues is given below.

```
/STATION/
NAME=*MP_ROAD.MP_RO_VQ;
TYPE=SERVER, MULTIPLE(2);
SERVICE= BEGIN
    CUSTOMER::Vehicle.MP_SC:= DISCRETE(ID_RO(1 STEP 1 UNTIL NB_R_OUT),
```
Vehicular Flow Simulators Based on Complex Queuing Systems

PRI(1 STEP 1 UNTIL NB_R_OUT));
WAIT(ROAD#(CUSTOMER::Vehicle.MP_SC).SAT);
V(INCLUDING(QUEUE)::MP_ROAD.MP_RO_RQ);
TRANSIT(ROAD#(CUSTOMER::Vehicle.MP_SC).STRETCHR);
END;

The station defines a multiple server with 2 servers. The service algorithm
assigns a destination road to a vehicle selecting it from a list of accessible
streets according to a given probability distribution. The service algorithm then
controls the state of the destination street through the WAIT procedure. This
procedure blocks the routing of vehicles when the destination road is saturated.
Otherwise, the service algorithm continues execution, releases the resource of the
road multiplexer and the vehicle is sent to the STRETCHR queue of the destination
street.

/STATION/
NAME=*MP_ROAD.MP_RO_RQ;
TYPE=RESOURCE,MULTIPLE(2);

MP_RO_RQ is a resource queue which regulates the access to the road
multiplexer. It handles transit requests of vehicles into the road multiplexer,
limiting the number of simultaneous accesses to no more than two.

5.5. The INPSEC object

In our simulation model this object functions as an input lane into an
intersection. It has two internal queue variables: INP_VQ and INP_RQ. Below we
report the code for defining the station templates for these two queues, along with
comments.

/STATION/
NAME=*INPSEC.INP_VQ;
TYPE= SERVER,SINGLE;
SCHED= FIFO;
SERVICE= BEGIN
CUSTOMER::Vehicle.ID_PAT:= DISCRETE(NCP(1 STEP 1 UNTIL NDIRECT),
   PR(1 STEP 1 UNTIL NDIRECT));
CUSTOMER::Vehicle.I:=1;
IF (SEM_FLAG <> 1) THEN BEGIN
   WITH CUSTOMER::Vehicle DO BEGIN
     WAIT(SEMAPH);
   END;
END;
WITH CUSTOMER::Vehicle DO BEGIN
   P(REQRT(ID_PAT),PRIORITY(ID_PAT));
   CST(SERV);
   V(INCLUDING(QUEUE)::INPSEC.INP_RQ);
   TRANSIT(VHERT(ID_PAT),PRIORITY(ID_PAT));
Each input section is associated with a number of paths for crossing the intersection, all starting at this input section. The service algorithm assigns to the vehicle one of the possible paths associated with the input section according to a probability distribution specified when data are inserted into the description file of the system [5]. The service algorithm then checks the `SEM_FLAG` variable to verify whether there is a traffic light system in effect. If so, the vehicle is forced to wait (using the `WAIT` procedure) for a green light before its servicing resumes. The continuation of the service inside the station template requires issuing an access request, through the procedure `P`, to the first inner section of the intersection that the vehicles is to traverse. Once the vehicle acquires the resource, the crossing time corresponding to its entry into an inner area of an intersection starts. Subsequently, the service algorithm involves the release of the resource acquired upon entry to the input section and the transit of the vehicle in the vehicle queue of the first inner section along its path for crossing the intersection.

```
/STATION/
NAME=*INPSEC.INP_RQ;
TYPE=RESOURCE,MULTIPLE(NB_VE);

INP_RQ is a resource queue of the input section. It controls the number of accesses to the input lane entering the intersection, which is limited to a maximum of `NB_VE` vehicles that can be simultaneously present. We can imagine that vehicles are queued in the input lane and are serviced one at a time by the `INP_VQ` station, according to the order in which they arrived, entering in sequence into the inner area of the intersection.

5.6. The SECTION object

This object is used to model the action of vehicles crossing inner areas of a street intersection. For simple intersections the inner sections have a capacity of 1, while for roundabouts the capacity of an inner section can be greater. This can be seen in previous figures illustrating single intersections in a street system, for instance in Figures 1 and 2. `VQ` and `RQ` are two internal queue variables. The definition of the station templates for these queues is given below together with comments.

```
/STATION/
NAME=*SECTION.VQ;
TYPE=SERVER,MULTIPLE(NB_V);
SCHED=PRIORITY,PREEMPT;
SERVICE= BEGIN
WITH CUSTOMER::Vehicle DO BEGIN
  CST(SERV);
  IF I <> PAT#(ID_INT,ID_PAT).L THEN I:=I+1;
P(REQRT(ID_PAT),PRIORITY(ID_PAT));
```
This is a queue with multiple servers. The number of servers corresponds to the capacity of the inner section and is given by the \( NB_V \) variable, whose value is set in the file describing the intersection, based on the physical structure of the section. Inside the section priority preemptive service scheduling is employed. This assumption is necessary in order to avoid deadlocks in the system. The service algorithm of the station template lets a certain time needed for traversing the station elapse, then issues a request for accessing the subsequent section in the path for crossing the intersection. Once the access request is granted, the service algorithm releases the previously acquired resource for entering the current section, subsequently routing the vehicle into the \( VQ \) queue of the next section in the path for crossing the intersection.

5.7. *The OUTSEC object*

In our simulation model this object reproduces the functionality of a single lane for exiting an intersection. The capacity of an exit section is equal to 1. An output section lets vehicles leave an intersection into a street. The connection between the intersection and the street is realised through the \( EXIT \) pointer. The internal queues of this object are \( VQ \) and \( RQ \). The station templates corresponding to these queues are defined below.

```plaintext
/STATION/
NAME=*OUTSEC.VQ;
SERVICE= BEGIN
   WAIT(ROAD#(ID_ROAD).SAT);
   CST(SERV);
   V(INCLUDING(QUEUE)::OUTSEC.RQ);
   TRANSIT(EXIT);
END;
```

The above is a station with a single server. The service algorithm uses the \( \text{WAIT} \) procedure to verify if the street to which a vehicle is to be directed is saturated. If so, the service blocks until space is freed in the street connection to
the output section. The service algorithm lets a certain time needed for traversing
the output section elapse, then it releases the resource previously acquired by the
vehicle for entering the current section, finishing with effecting a transit of the
vehicle along a street that is an exit from the intersection.

/STATION/
NAME=*OUTSEC.RQ;
TYPE=RESOURCE;
SCHED=PRIOR, PREEMPT;

RQ is a resource queue which regulates accesses into the output section.
In this case this is a single resource, which ensures that only a single vehicle can
transit through an output section at a time. This mechanism allows the simulation
to reproduce the potential congestion effects that may take place at the entry to
a street that is an exit from an intersection. Priority preemptive scheduling is
used to ensure deadlocks do not take place in the system.

5.8. The SEMCOMP object

In our simulation model this object represents a traffic light (semaphore),
which, using a green or red light, respectively, allows or blocks access to an
intersection from a given input lane. In the construction of the simulator this
component will thus be connected to an input section. If an intersection is
signalised, each input section will be associated with a semcomp object. Such an
object is a system of queues traversed by a single control signal which is a client of
the csc class. The internal queues of this object are R1, R2, G1, G2, Y. The station
templates constructed for these queues define the structure of a control circuit
of a generic semaphore component. The code for defining it is given in the block
below.

/DECLARE/
CLASS CSC;

/STATION/
NAME=*SEMCOMP.R1;
INIT(CSC)=FR1;
SERVICE=CST(TR1);
TRANSIT=R2;

/STATION/
NAME=*SEMCOMP.R2;
INIT(CSC)=FR2;
SERVICE=CST(TR2);
TRANSIT=G1;

/STATION/
NAME=*SEMCOMP.G1;
The green and red phases of the semaphore are subdivided in two sub-phases, respectively G1, G2 and R1, R2. Y represents the phase of the semaphore corresponding to yellow. One of the variables FR1, FR2, FG1, FG2 shown in the definition of station templates above can have a value of 1, while the remaining variables must be zero. In this way the system formed from the phases of the traffic light cycles a single control signal. As the signal enters in one of the phases R1, R2, G1, G2, Y, it activates the corresponding traffic light, and thus blocking or allowing vehicle flow into the intersection from the input section it is associated with. In this way the operation of the traffic light for each input section is managed, while keeping the components mutually independent. At the beginning of the simulation the phase of the traffic light corresponding to the station template which has been initialised with an internal control signal is activated. Then, the operations defined in the active station are executed, after which control proceeds sequentially through subsequent stations. In the service algorithms of queues G1 and G2, RF(ID) is a pointer to a variable controlled by the service algorithm of the INP_VQ queue of the input section connected to a traffic light component, through the WAIT procedure. In this way, when this variable is set to closed, the WAIT procedure results in blocking the vehicle in service and the input flow through the section is stopped until the state of RF(ID) is reported as open. The service algorithms G1 and G2 ensure that the state of RF(ID) is open, maintain this state for the duration of the green phase of the traffic light, and subsequently
set the state of RF(ID) to closed. Thus, the control signal traverses the queues G1, G2, Y, R1, R2, activating, respectively, two green phases, one yellow phase, and two red phases.

6. Conclusions

In this work we presented an object library which we use for constructing simulators of urban vehicular traffic flows. Such systems are formed by street intersections mutually connected by urban streets. The operation of street intersections can be controlled by means of traffic lights. A particular type of street intersection is a roundabout.

In this work we aimed to present a system formed by two roundabouts and two non-signalised intersections. In this way we were able to illustrate the specification stage of the system, carried out by means of objects defined in the library. We showed how by using a reference map it becomes possible to associate particular components of a traffic system with instances of object types defined in the proposed library that simulate their functionality.

In our future work [5] we will aim to illustrate how, basing on the above approach to specification, it becomes possible to construct a file describing a street system. Thus, given a street system we will be able to associate a description file and subsequently define a procedure that will automatically generate a simulator of the traffic system in question.

References

[1] Pasini L and Feliziani S 2010 TASK Quart. 14 (4) 405