Real-time 3D Visualization for Distributed Simulations of VANets †

Shichao Guan, Robson Eduardo De Grande, and Azzedine Boukerche

PARADISE Research Laboratory
School of Electrical Engineering and Computer Science
University of Ottawa, Canada
Email: {shichao,rdgrande,boukerch}@site.uottawa.ca

***Abstract*—Evaluation and validation of algorithms and protocols in vehicular area networks is challenging and requires the support of simulators in most cases due to the restrictions on cost and scalability. Consequently, there is a need to identify or build a simulator that best fits into the characteristics of VANets. Such simulators need to reproduce the communication of networks together with the mobility of vehicles in a given simulated area. Many simulators and simulation frameworks have been developed, most of them combining pre-existing mobility and networking simulators in one solution. However, these simulators present limited features on 3D visualization. In this paper, we propose a real-time, realistic 3D visualization for VANet simulations, which makes use of 3D-modeled real-world maps; the proposed system effectively generates the intended visualization. Experiments have been conducted to evaluate the performance on the synchronization between simulation and visualization components through an analysis of overhead and delays.**

I. Introduction

Vehicular Area Networks (VANets) have been receiving a growing attention lately due to interests in developing solutions for traffic management and control, public transportation, safety, content delivery, and many others. Such solutions require the use of new techniques and concepts in this area, such as context awareness and object detection, or it needs modifications or improvements on the existing routing protocols, service discovery mechanisms, and coverage and localization techniques. All these new solutions are designed based on different types of communications that range on vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-sensor interactions.

Due to this increasing number of works on VANets, a need for evaluation and validation also grows. The evaluation of new solutions is necessary for determining precision or effectiveness of a given technique, acceptance by the community and industry, and the development of standards to be finally implemented in real-world. For most cases, real experiments or the construction of prototypes for validation are extremely costly due to the nature of experimental scenarios; such scenarios are composed of a large number of objects (vehicles) for large-scale environment matching real cities, they must be evaluated on urban and road environments to observe the performance of evaluated solutions with real-world obstacles, or they have to be evaluated extensively during a long period of time for realizing a detailed analysis on performance and flaws. Consequently, simulations are used for extensive evaluation and can be an effective tool for verification before implementing new techniques on real scenarios.

Several network simulators, or simulation frameworks, have been developed for testing new protocols applied on different layers of the networking stack, such as on MAC layer, routing layer, transportation layer, or even on a cross-layer. On the same way, several mobility, and traffic control, simulators have been designed for observing traffic behavior in urban areas or roads upon the use of new techniques or for managing traffic efficiently. Besides these two types of simulators, some other simulators have been created for joining the mobility and networking simulations to be properly applied on VANets. Most of these simulators are established as frameworks that interfaces the communication between a mobility simulator and a network simulator, which lack some precision of details depending on interface methodology between the parts; a few of them really realize both vehicle movement and networking simulation together, not needing a third-party simulator. However, all of them lack a real-time, realistic 3D visualization of their simulation outputs, which is useful for observing microscopic details on events in particular simulation incidents.

We proposed a simulation system that combines simulation and visualization in this paper. The proposed simulator aims at introducing a realistic visualization of simulated objects during run-time. This visualization requires 3D modeling of environments based on real maps collected from on-line sources. A sophisticated gaming and physics engine is incorporated in the proposed system to increase the realism of the simulations and to provide detailed, microscopic vehicle movement simulation and visualization. A distributed simulation framework is also used in the system to coordinate the distributed execution of simulations. We also show in our experiments that the proposed solution is able to handle the simulation and visualization properly up to a given number of objects; for a large number of objects, modifications on the gaming engine used in the system are needed to cope with processing overhead.

This paper is organized as the following. Section 2 presents the related work and highlights the challenge issues. Section 3 introduces the system by describing its architecture, components, and functioning. Finally, the conclusion succinctly summarizes the paper and presents the future work of the ongoing work.

† This work is partially supported by NSERC, the Canada Research Chair program, ORF funds, and EAR Research Award.

II. Related Work

Due to the constant need of simulators that can join mobility (road traffic) and networking (wireless communication), several solutions have been developed aiming at enabling simulations of vehicular area networks. These solutions, the great majority of them, in most cases, are based on a network simulator that incorporates mobility models required for guiding the movement of the simulated objects. Consequently, all the solutions listed in this section present two major parts, which consist in producing the mobility of moving objects, vehicles, and statically or dynamically using this mobility output for the execution on a network simulation platform.

Vanet Mobility Simulation Environment (VANetMobiSim) [1], [2], [3] is a mobility simulation environment which is specifically designed for vehicular mobility at both macroscopic and microscopic levels, observing realistic automotive motion models. Being Java-based, VANetMobiSim accepts maps in the Geographical Data File (GDF) standard format and implements many mobility, physics, and vehicular dynamics models; the mobility simulation tool can generate movement traces to support different mobile network simulation tools. Consequently, this tool is restricted to only generate mobility traces of moving objects in imported that can be later used in network simulators. Some tools, such as HWGui [1], can help on providing a simple, restricted visualization of the movement of objects for auditing and evaluation.

Similarly to VANetMobiSim, Simulation of Urban Mobility (SUMO) [4], [5] is a road traffic simulator designed, mostly, by the Institute of Transportation Systems at the German Aerospace Center to support large road networks. This simulator provides mobility simulation at a microscopic level through a open source, portable code. The portability and extensibility of the software enables it to interoperate with other applications at run-time, enabling network simulation over the movement of objects dynamically generated by SUMO. Some simple visualization interface are implemented for this simulation tool; they are used for verifying the deployment and execution of urban mobility simulations. Its light-weight execution and interoperability promotes the use of several network simulator that access its object movement output dynamically or statically.

Scalable Wireless Ad hoc Network Simulator (SWANS) [6], [7] is a wireless, and sensor, network simulator built on top of a Java in Simulation Time (JIST) simulation platform. JIST is a general-purpose discrete-event simulation engine based on Java language. This simulator aims at high-performance and efficiency. JIST/SWANS has been developed to fulfill the needs for network simulator, emphasizing on scalability: larger number of simulated objects and throughput. Even though of its simulation capabilities, this simulator requires an external, third-party tool that provides a mobility model for its objects when simulating vehicular areas networks.

OMNeT++ [8], [9], [10] is a discrete-event simulation framework specially built for performing network simulations. Since network simulation comprises wired and wireless networks, this simulation tool can be used to simulate IT systems, queuing networks, and hardware architecture. Together with the simulation framework, an object library is provided to facilitate the design of simulations through reflection, modularity, open data interfaces, reusability, and embedding support. Due to its modular architecture, it allows extensible and customizable network simulations, which enables dynamic modifications on network topology. However, OMNeT++ by itself, does not present a mobility simulation, so it requires a third-party simulation tool for integration.

Network Simulation 2 (NS) [11], [12] is another discrete-event simulator focused on networking; it has been widely used on validating tools, protocols, and solutions which are related to networking research. This simulator is a result of a joint work and efforts from many collaborators in order to present simulation of wired and wireless networks with considerable support on the existing, and well-established, protocols, such as TCP, routing, and multicast. NS provides animation tool called Nam [2], which is based on Tcl/TK [3] [4]; this tools uses the traces of NS as input to generate simple, limited, off-line visualizations of the execution of the simulations. Since this simulator is totally focused on networking, it requires object movement traces from mobility simulators.

Traffic and Network Simulation Environment (TraNS) [13], [14], [15] is tool developed for integrating traffic and network simulators, more specifically, SUMO and NS. Due to the need of NS for mobility trace files to coordinate the movement of simulated objects, many solutions have employed unrealistic VANet simulations, which are restricted to a predefined path. TraNS, on the other hand, introduces a mechanism that allows the network simulation modify the mobility of objects through an interface between the two simulators, SUMO and NS. The main part of this interface is TraCI [16], [17], which works as coupling interface that allows the network simulator to give atomic commands to the road traffic simulator, altering the movement of particular objects. The visualization interfaces offered by this tool are then limited to the NS and SUMO.

Vehicles in Network Simulation (Veins) [18], [19], [20] is a VANet simulation framework focused on vehicle-to-vehicle communication. This framework is based on two widely-used simulators, SUMO and OMNet++. Veins joins these two simulators by coupling them together bidirectionally, so updates on objects' movements are considered by updates from the Road Traffic simulator, SUMO, and the Network simulator, OMNet++. Through the use of TraCI, Veins performs the interfacing between the two simulators, which run in parallel. Consequently, its visual interface for monitoring simulations is inherited from SUMO and OMNet++, which present views restricted to verification and evaluation.

Opportunistic Network Environment (The ONE) simulator [21] provides both the simulation of networking and mobility; however, the simulator is built emphasizing on delay-tolerant networks (DTN). Besides simulating mobility of objects, the ONE accepts traces from other mobility simulators, such as SUMO. The simulated communication in the ONE embraces several DTN routing algorithms and different types of senders and receivers, restricting the use of this simulator to DTN-related scenarios. A simple graphical interface is provided for

[1] http://pi4.informatik.uni-mannheim.de/pi4.data/content/projects/hwgui/

[2] http://nsnam.isi.edu/nsnam/index.php/Nsnam:About

[3] http://www.tcl.tk/

[4] http://wiki.tcl.tk/

real-time verification.

In the same line for work, another framework has been introduced in [22] to join OPNET [5] [6], a commercial network simulator, and SUMO. The proposed framework has been developed for simulating VANets, and it generates a output file, as a result of the simulation, or it shows the result on a graphical interface through the positions of the objects.

A work described in [23] also combines both mobility and networking simulation. This simulator, especially built for VANets, presents a complete integration of mobility and networking, enabling a scalable solution. The visualization of the proposed simulator shows the output of the simulations effectively; however, the system describes a limited 3D visualization interface, which is simple and lacks realism.

Based on the existing works built or suitable for the simulation of VANets, there is no solution that can provide a real-time 3D visualization of the simulation output. This visualization can enable the analysis during run-time as an immersion virtual environment; it also can allow the coexistence of other systems that might behave as a visualization tool but feed the simulation system with up-to-date real data. In order to provide such a solution, issues involved with synchronization, responsiveness, and data abstraction need to solved; all these aspects are directly related to guaranteeing real-time coordination and 3D presentation of simulated objects.

III. Proposed System

Observing real-time 3D visualization of VANet simulations as the main focus of the proposed system, the design of the system is totally oriented towards the use of and synchronization with tools for 3D visualization and manipulation in its first development phase. As a result, the system is intended to join distributed simulations and third-party tools for 3D visualization, virtual environments, and gaming, introducing a modular design since the beginning of its conception. The flexibility of a modular design allows the system do use different simulation infrastructures, as well as a variety of tools or engines for 3D visualization. As a consequence of this decoupling design components, additional synchronization mechanisms are required to integrate and coordinate the execution of simulations. This synchronization is needed for runtime object updates and the initial setup and positioning of the map used in the system. Finally, real-world maps are intended to be used in order to run simulations as realistic as possible, mimicking actual places and regions.

A. Architecture

As shown in Figure 1, the architecture of the proposed system is divided into two main parts: visualization and simulation. Since both parts of the proposed system are interdependent, they required constant synchronization. The communication between visualization and simulation is performed through components delimited by the area in the dotted line in the figure; the whole communication is performed through interfaces, Visualization Communication Interface and Simulation Communication Interface. These two components are responsible for buffering and transmitting data to the endpoint, which might be located in the computer or remotely. A parser is also used in such interface in order to translate the simulation updates into the proper visualization format, as well as the reciprocal communication.

For the synchronization between parts, the Mapper in the simulation side of the architecture is essential for keeping the visualization object updates consistent with the information the simulation contains for the positioning of all simulation objects. This component is responsible for matching the position and simulation and visualization versions of the same objects. In the visualization side of the architecture, the Update Feedback Daemon runs in parallel with the visualization components, and it receives, buffers, and makes available the incoming updates regarding the visualization objects. The daemon is critical for no disrupting the execution of the 3D Visualization Tool. Upon the receipt of messages, variables are allocated with their respective new data, which is used they tool to update or modify the coordination of objects movements.

1) Visualization Components: Two main components, intrinsically related and connect the third-party visualization tool, perform most of the actions for microscopic transformations in the visualization and correctly displaying simulation to the user: Physics Engine and the Renderer. The physics engine particularly provides means for defining and controlling the microscopic actions for the movement of objects. The engine is essential in enabling physical transformations in the environment, so it is directly related to performance and on the quality of the resulting output for the visualization of the simulation. The render is basically responsible for dynamically rendering objects and structures in the environment. As a matter for performance improvement, only the objects in the line of sight are rendered even though the rest of the objects, not being visualized, have their movements coordinated and their status controlled. Together with the Renderer and the Physics Engine, the Controller, which is not shown in Figure 1, coordinates the transformations on the objects, using the Physics Engine, as well as the Renderer for the final result. The Controller is also called for passing updates on positioning of objects to coordinate with the used 3D map.

The 3D map not only contains the map which is used for placing the objects on but also it encapsulates a Map Controller. The Map Controller is responsible for providing the map synchronization in the startup of the simulation the checking the correct positioning of a object. During the simulation, the controller is mostly called for incoming updates that need to be verify before passed to the controller of each object for movement coordination.

The 3D Objects concerns all objects being manipulated by simulation and in the visualization, as well as the Object Controller. Each 3D Object contains all data related to a object, which essentially consists in its positioning, direction, way points, visibility, and type. The Controller is related to the incoming updates on positioning, short-term movement coordination commands, and current status of the object, as well as its consistency in the visualization with the simulation.

2) Simulation Components: The Map on the simulation side of the system contains the coordinates that define the map used to control the movement of objects during the simulation.

[5] http://en.wikipedia.org/wiki/OPNET
[6] https://www.opnet.com/

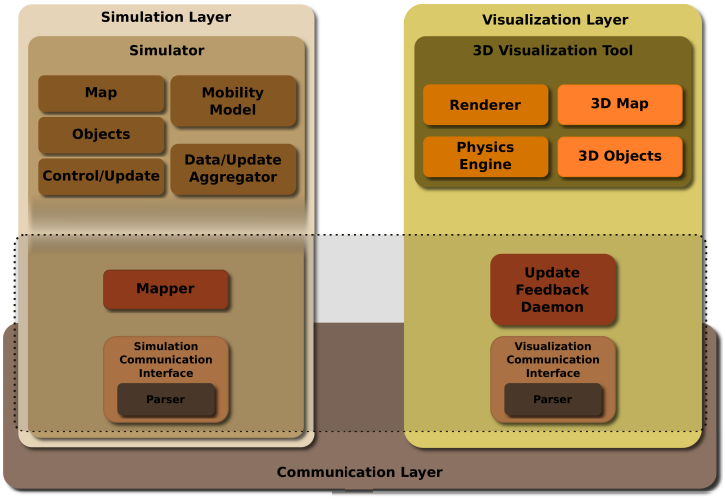

Fig. 1: General Architecture of the Simulation/Visualization System

These coordinates are store in a linked list and in the form of a set of edges and points that define a directional graph. Together with the data structure that contains the coordinates, a controller is also used to manage the positioning of objects into a edge of the graph; the management consists in first verifying the consistency of the synchronized positioning information of an object and linking objects and structures for facilitating the identification of positioning for the simulation.

The simulation Objects contain data structures that reflect the positioning of 3D Objects in the visualization tool. Such data structures also can contain further data that represents the object and might be useful or essential later for determining the status of a simulated object, as well as its position in the simulated map. Together with the data structure, methods are part of Objects component. These methods are responsible for triggering the updates on positioning for later being passed to the visualization; basically the methods call the controller in order to access the Mobility Model defined for the simulation. Upon on the decision-making for the step ahead in the short-term future movement, these methods return new coordinates.

The Mobility Model component contains a selected mobility model for coordinating the movement of objects on the map. The model is consulted by each Object, and it accesses the map in order to retrieve information about the region nearby an object, represented by the neighbouring edges in the map graph. The access to map grants a so-called global view to the Mobility Model component.

The Control/Update component is called by the Mapper whenever an update on the positioning of objects is needed, so it works responsively upon the receipt of calls. This component triggers the Objects component for obtaining an actualization on the coordinates of a given object. When the resulting new position is retrieved, it is passed to the Data/Update Aggregator. The Data/Update Aggregator is responsible for collecting messages together to be sent to the visualization components over either through message passing communication or over the network. In order to decrease the frequency and the amount of data transmitted, all updates are aggregated by this component.

B. *Functioning*

Similarly to the concepts that drive the design of the system's architecture, the whole proposed system presents two major parts that run in parallel: the visualization and the simulation. In this case, synchronization is needed for keeping the execution consistent in both parts. The synchronization occurs since the startup of a simulation and endures until its end, and it is performed through shared memory or over the network. Due to this division on roles between simulation and visualization, basically elements existing in one of them need to be replicated in the other, being constantly updated for simulation consistency purposes.

Before the simulation starts, a preparation needs to be performed, as shown in Algorithms 1 and 2. In both algorithms, at set of procedures are required to be followed in the startup of both parts of the system. These procedures guarantee that the positioning between maps, simulation and visualization, as well as objects, are uniformly matching before the simulation starts.

For the initialization of the simulation, a map is loaded, and then it is synchronized with the visualization. The map synchronization consists in retrieving references of the map, which is described in line 2 of Algorithm 1 and line 2 of Algorithm 2, and using them to rotate, translate, and scale the simulation map, as presented in lines 3, 4, and 5 of Algorithm 1. After the maps are matched, this initial transformations on the simulation map are concluded, avoiding later calculations for translating incoming visualization 3D object coordinates into simulation objects and speeding up the response time on updates.

As a second part of the initialization, the simulation also needs to instantiate objects that represent vehicles or pedestrians in the simulation. The number and type are defined in configuration input for the simulation. After these objects are created and deployed, as shown in lines 7 and 8 of Algorithm 1, they are synchronized with the visualization components, as described in line 9 of Algorithm 1 and line 3 of Algorithm 2. The synchronization consists in exchanging data about the new objects created in the simulation, creating

Algorithm 1: Simulation Execution Algorithm

```
1  Require: map_vector, obj_definition, MobModel /* Initialization
       Phase                                                    */
2  get_sync_map()
3  map_sim ⇐ rotate(map_sim, map_3D.references)
4  map_sim ⇐ translate(map_sim, map_3D.references)
5  map_sim ⇐ scale(map_sim, map_3D.references)
6  match_maps(map_3D, map_sim)
7  objects_sim[] ⇐ create_sim_objects()
8  deploy_objects_map(objects_sim[], map)
9  sync_visualization_objects(objects_sim[])
   /* Execution Phase - Macroscopic                            */
10 while !sim_end do
11     if update_request_arrive(obj) then
12         edge ⇐ find_edge(obj)
13         if edge == ∅ then
14             obj.pos ⇐ redeploy(MobModel, obj.pos)
15             edge ⇐ find_edge(object)
16         else
17             new_edge ⇐ move(MobModel, edge, obj.pos)
18         end
19         obj.wayPoints[] = get_wayPoints(new_edge)
20         send_updated_data(obj)
21     end
22 end
23 send_end_signal()
```

Algorithm 2: Visualization Control Execution Algorithm

```
1  Require: map_3D, obj3D_definition /* Initialization Phase   */
2  sync_simulation_map(map_3D.references)
3  get_sync_objects()
4  objects_3D[] ⇐ create_objects()
5  deploy_objects_map(objects_3D[], map)
   /* Execution Phase - Microscopic                            */
6  while ∃sim_execution do
7      for obj_3D IN objs_3D[] do
8          obj_3D.pos_x,y,z ⇐ move(obj_3D.pos_x,y,z)
9          obj_3D.wayPoint ⇐ updateWP(obj_3D)
10         if need_update(obj_3D) then
11             send_update_request(obj_3D)
12         end
13     end
14     if receive_obj_update(obj) then
15         obj_3D ⇐ update_obj(obj)
16     end
17 end
18 shutdown_visualization()
```

them in visualization environment, and deploying them on the 3D map in their respective positions, as described in lines 4 and 5 of Algorithm 2. In the simulation side of the system, the deployment defines the initial mobility commands for these objects in the visualization part of the system, which is needed to initiated on the microscopic execution of the simulation.

Please note that without this map initialization and object matching in the startup between the simulation and visualization parts of the system, neither of these processes continue on running. Consequently, the execution of the simulation is triggered by the visualization components after the synchronization is individually finished by each 3D object.

As shown in line 10 of Algorithm 1 and line 6 of Algorithm 1, the execution of both parts of the system are represented as loops but not necessarily being designed as such; the loop in both cases represent the constant, timely checking for and responding to object updates. Thus, the simulation runs while it is required, and it sends an *end signal* to the visualization components when it is complete, as shown in line 23 of Algorithm 1. The visualization keeps running until it receives a signal to terminal, stopping completely and shutting down, as described in line 18 of Algorithm 2.

As represented in line 7 of Algorithm 2, the visualization constantly runs, updating the position of objects and rendering them on the map. The Controller of the visualization part uses the method *move*, in line 8 of Algorithm 2, to update the positioning of each object in a microscopic scope for providing a smooth visualization of objects on a given simulation scenery. The method move consists in checking on the current status and position of an object and calculating the new position based on the predefined current way point of the object; this calculations employ scripts that define the motion of real objects, vehicles, based on a physics engine for producing realistic movements.

The base structure on the visualization for determining a realistic movement of objects makes use of way points to define short-range paths. According to this mechanism, an object necessarily requires a way point to follow at any time. In case an object reaches its way point and needs to continue moving, it follows the next way point in the long-range path. Thus, the movement control scripts checks the status of an objects, as well as its proximity to the followed way point, and re-assigns a new way point to the object in case needed, as shown in line 9 in Algorithm 2.

In case the system detects that there is no next way points for a given objects in the following swap of way points for the movement of the object, the systems preemptively reacts and requests the simulation components an update on the way points for the 3D object. This is represented by lines 10-11 of Algorithm 2. It is worth noting, that since the system performs this actions preemptively, the object does not suspend its movement; the suspension in the movement of the object might occur only if there is a long delay on response time from the simulation interface components. As shown in lines 14-15, upon on receiving an answer from the simulation on the requested update on the way points of an object, the visualization controller updates the object with the new received way points. In case, a mismatch between the positioning of a 3D object and its respective simulation object, the simulation sends new coordinates for redeploying the object; these coordinates represent a location on the map closest to the synchronized position of the 3D object.

On the simulation side of the proposed system, the controller is constantly waiting for update requests, which are interpreted, evaluated, calculated, and promptly answered back to the visualization interface, as represent by line 11 in Algorithm 1. Consequent to the receipt of a request, the controller identifies the edge related to the position of the given object, as shown in line 12 in Algorithm 1.

In case the object cannot be assigned to any edge in the map due to mismatch, calculation error larger than accepted, or error in the microscopic movement of the 3D object by the visualization controller, a new edge is identified and assigned to this object, as present in lines 14-12 in Algorithm 1. The process of identifying an approximate edge consists in using the current position of an object and identifying the closest edge to it and projecting the object's position on the edge, which is called *redeploy* in the pseudo-code; all this process is completed considering that conflicts with other objects need to be avoided in case many objects are already movement

on the chosen edge. After these steps, the object related to the incoming request presents its position fixed and can be processed in the next step in identifying an update for it.

In the other case that an object is successfully assigned to an edge, the controller identifies the next edge, as shown in line 17 in Algorithm 1. The process in identifying the next edge follows a macroscopic scope in the movement of objects; it requires to access a mobility model, which might already have calculated all or some of the future macroscopic positions of the object or might be calculating the position on demand. For the design of this proposed system, the mobility model uses an on-demand approach in order to be more flexible and allow the input of real-time traffic data in the system later on.

After obtaining the edge for the object in this update request, the simulation controller just determines the way points related to the future position of the object, or the current position in case the object needed to be redeployed, as described in line 19 in Algorithm 1. As the way points are identified, they are assigned to the object and then sent to the visualization interface.

At this initial stage of design of this system, the simulation only controls the objects mapped in the visualization, which can be observed in the simple pseudo-code of Algorithm 1. However, besides controlling such objects, the simulation also is expected to coordinate other elements, which are indirectly related to and influencing the objects being visualized. As a further extension of these system, the simulation is expected to control part of the simulation objects movements in a microscopic scope too; this measure is used as a tool for harnessing the performance benefits of parallel and distributed simulations, alleviating part of the burden from the visualization components and allowing truly scale up of the simulated system.

C. Design and Implementation

The design and implementation of the proposed system is based on using a set of third-party tools, which are merged with the simulation through components that facilitate the integration of both parts. The third-party tools are used mostly for the design of visualization components of the developed environments. The use of third-party also compelled this design specifically, which is constituted of two may parts, visualization and simulation.

The methodology for enabling the system consisted in generating a procedure for obtaining and inserting real maps into the visualization/simulation parts. This required the use of openly available maps, which can be easily retrieved from on-line source; with this objective, *Open Street Map* [24], [25], [26] is used for its accessibility and its features: to be build based on local knowledge, enabling more accurate and up-to-date maps; to be driven by the community, accepting public GPS traces; to present open data, having maps under the ODbL format; and to be able to export its map data to the standard Global Information System (GIS) data, consisting in a format of OSM XML.

The obtained map in standard format is a graph constituted by a set of edges and vertices. In order to have this map suitable for 3D visualization, it needs to be populated with 3D elements to compose the landscape of a particular area. Thus, Esri City Engine [27], [28], [29], a third-party tool for 3D urban design, is employed to populate the plain map. Besides accepting 2D GIS data and exporting 3D city models, this engine is widely used for urban planning, architecture, and design, it uses and incorporates realistic context, and it allows sketching and texturing of 3D building models. With the concept of zoning, the engine populates an area of a map with similar buildings, following a given pattern; this feature allows the system to reflect realistic, but not real, aspects of the landscape of a map. Moreover, Autodesk 3DS Max [30], [31] is used on the generated 3D map for modifying or inserting elements and for adjusting center of mass of already-defined landscape elements. This latter third-party tool presents powerful mechanisms for 3D modeling and animation: vector map support; mesh and surface modeling; texture assignment and editing; and shading and material design. The tool thus provides means for inserting 3D objects specifically designed for a given map to represent reference points from a particular region, such as towers, statues, or even buildings. At the end of the process, which is described in details in the next section, the 3D is ready to be incorporated into the visualization components.

The implement of the visualization part of the system is then concluded with the use of Unity 3D [32], [33]. Apart from being able to import generated 3D maps from other tools, this engine is a powerful game design platform, which presents fast rendering mechanism for execution and compatibility with many platforms and coding languages; it uses a industry-standard physics engine, presents optimization on performance through real-time shadows, baked lighting, particle system, post-processing effects, pre-computed occlusion culling solution. In this game platform, the other visualization components are coupled to connect the execution engine together with the simulation components, consisting in the Visualization Controller, Update Feedback Daemon, and Visualization Communication Interface.

On the simulation side, federates are built following the HLA platform [34] and making use of the HLA Run-Time Infrastructure (RTI) for coordinating the execution. The HLA RTI implementation used in this design is the Portico RTI [35], which provides open source, multi-platform solution. HLA was devised to coordinate the execution of distributed simulations and facilitate their design, aiming at re-usability and interoperability. A HLA simulation, called federation, is basically composed of federates coordinated by management services, which are provided the RTI. In the scope of the design of the proposed system, one federate is built to contain the Simulation Communication Interface, Mapper, and Data/Update Aggregator. This federate is responsible for translating and sending the incoming update requests to the rest of the simulation using the HLA communication middleware to consistently feed the simulation with up-to-date positioning data. The rest of the federates share the map used in the simulation and the mobility model. Simulation objects are split among the federates following a predefined pattern, such as enforcing performance; the federates coordinate the movement of their respective objects by communicating with other federates, as well as accessing the map and the mobility model. The communication among federates is performed through publish/subscribe methods; this communication also involves

the transfer of updated object positions to the visualization interface federate.

D. Map Processing

The environment is the major part in the system to support a realistic visualization; more details from the real scenery increase the realism, as well as the complexity in designing the environment. Since the whole the scope of the simulations dealt by the proposed solution consists in urban and/or road scenarios, the map that constitute the simulation comprises the whole environment or most of it. As a result, the map processing requires extra attention in providing accurately realistic presentation of objects, which attempt to mimic the details of elements on the landscape. In this section, we present the major steps needed for producing such maps.

The first stage of map processing consists in obtaining the map, which is retrieved on-line through *OpenStreetMap*. At this point, the map is just a graph composed of a set of edges and vertices as shown in Figure 2 a), which represents the city Ottawa, Canada. Even though precise and consistent with real streets and roads, there are no objects that belong to the landscape of the map. The map at this stage is functional, so the simulation/visualization can be executed with the map at this point; however, the map lacks realism. Consequently, the next stages performed to add virtual landscape elements.

The second stage comprises populating the map with buildings. This stage can be repeated iteratively, as many times as it is required to insert the different layers/categories of buildings that are supposed to compose the map. The task of populating the map is totally performed by using the CityEngine tool, which allows to populate a whole selected area of the map at once. Note that the use of this urban design engine, or any other similar tool, is not needed to populate the map, but the feature of populating regions of the map with similar buildings considerably speeds up the process. The tasks realized in this stage are depicted in Figure 2 b) and c), which distinctly shows two categories of buildings being inserted.

The third stage involves adding distinguished landscape objects on the map. These objects can be prominent or famous buildings that characterize the scenery of a region. Autodesk 3DS Max is used to generate these objects and add them to the map. For instance, as present in Figure 2 d) and e), which respectively shows the buildings representing the Canadian parliament buildings and adding them to the Ottawa city map.

Finally, with the whole virtual map defined, textures are added to the map and 3D objects. The last stage of the map processing is depicted in Figure 2 f). At this point, the map is ready to be loaded at the gaming engine for synchronization and execution. Similarly, the map obtained at the first stage is loaded at the simulation part of the system for execution.

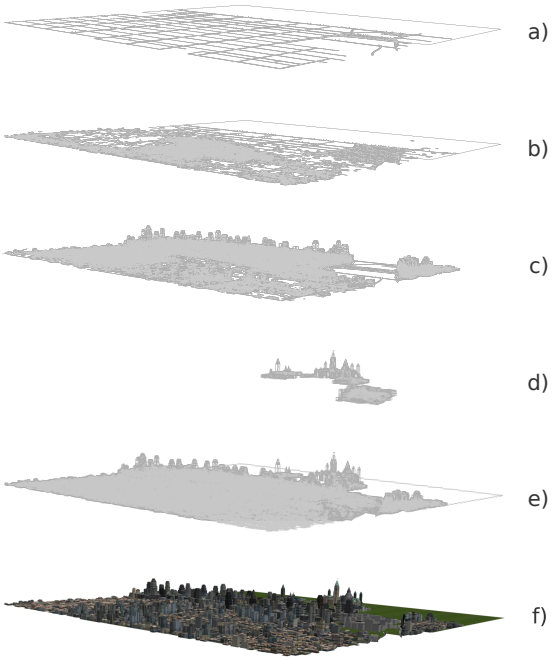

Fig. 2: Process line of obtaining a full texturized 3D map

IV. Experimental Results

Experiments have been conducted in order to evaluate the proposed simulation/visualization system. Through the experiments, it has been possible to observe the overhead caused by the synchronization between both visualization and simulation components. A scenario has been defined for the experiments; the scenario consisted in using a map representing Ottawa downtown, as shown in Figure 2 with a growing number of vehicles. At this stage of analysis, a naive, random mobility model has been used. The visualization and simulation components have been deployed on two computing servers connected by a gigabit ethernet link. Each computing server consisted in a Intel(R) Core(TM) i7 920 2.67GHz composed of 8 cores, 8 Gigabytes of RAM capacity, and an AMD/ATI Radeon HD 4850 graphics card. Three analysis scenarios have been realized: communication delay and overhead, processing overhead, and visualization performance.

In the analysis of communication delay and overhead, the incoming and outgoing communication data has been collected in the computing server hosting the simulation components. As shown in Figure 3, a curve represents the incoming packages and the other the outgoing packages. As the number of simulation objects grows, the overhead on the communication, bandwidth consumption, increases. There is a difference on bandwidth consumption between the two curves of of around 20 kilobits per second; this difference is almost a constant for any number of simulated objects. The reason this difference might be partially caused by the background communication load from the server and by the difference on packets issue from the simulation components and from the visualization components, which has not influenced much the consumption, otherwise there would be a steeper slope on the curve representing the incoming packages. Note that the curves increase the bandwidth consumption as the number of objects grow, but the curves tend to stabilize as the number of objects reach 130; this behavior is caused by the influence on computation overhead, which is discuss in the analyses.

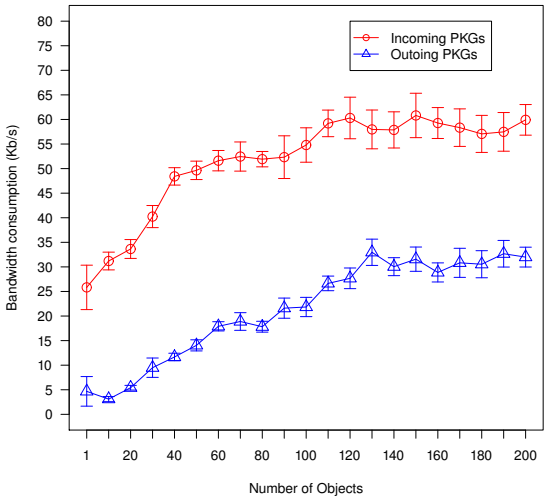

Fig. 3: Incoming and Outgoing Bandwidth from the Simulation Interface Component

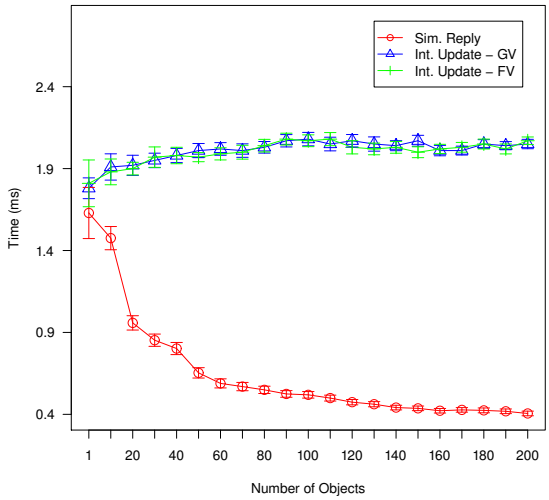

Fig. 4: Processing Overhead from Object Updates

The analysis of processing overhead consisted in observing the amount of time required for simulation components to process the incoming requests on data updates and for the visualization components to update the 3D objects with the incoming data. As shown in Figure 4, the graph presents three curves, one for the simulation processing times, and two for the visualization times; for the visualization, two approaches have been used in order to measure performance considering rendering times of the gaming engine: global view (*Int. Update - GV*) and focused view (*Int. Update - FV*). The global view shows on the screen the view of whole map and focused view presents a view from a camera that shows only one block of the city map. According to the features of the gaming engine used in the implementation of the proposed system, the engine only renders the 3D objects contained in the view shown on the screen for performance gain reasons. As presented in the figure, both curves that represent the processing of visualization components show very similar times; both curves present a slight increase in the times, and they are always higher than the curve representing the simulation processing time due. On the other hand, the curve related to the simulation response time (*Sim. Reply*) decreases its times as the number of objects increases. This behavior is caused due to the manner that the times are obtained and the message buffering technique. This reply time comprehends the whole time for generating updates and sending them back to the visualization components. In this case, the buffering of messages in the simulation components delayed the response time for the context of low number of object updates, which can be clearly observed in the graph.

The last analysis on these experiments concerns visualization performance, which is measured in frames per second (FPS). As described before, this analysis compares the rendering performance using a *Global View* and a *Focused View*. The global view is expected to consume more processing time due to the existence of a larger number of objects, vehicles and buildings, to be rendered. This larger overhead can be observed on the difference of FPS produced by the gaming engine. As the number of objects increase, the difference between the two

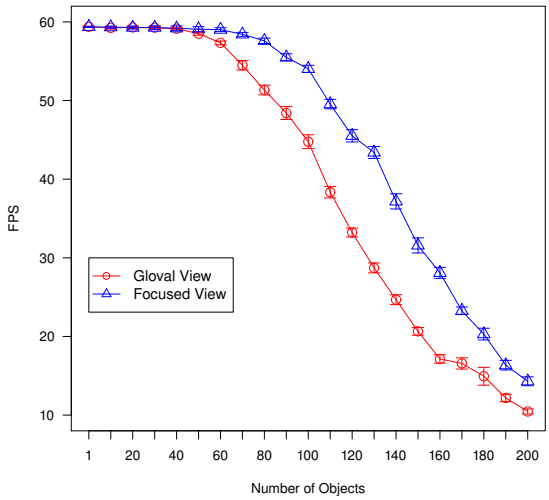

Fig. 5: Rendering Performance of Visualization Interface

curves also grows. This means that the feature of the game engine for a focused view considerably improves performance on rendering. However, the difference starts decreasing with a number of objects larger than 170; this is originated from the increasing overhead on controlling the microscopic movement of vehicles. The overhead on control with a large number of objects also influences the communication overhead; since the visualization interface spends more time on controlling and rendering, it requests less updates from the simulation components. Therefore, the control on vehicle movements presents the bottleneck in the system, and modifications on how this control is performed are needed for enabling large-scale simulations.

V. Conclusion

We have presented in this paper a realistic, real-time 3D visualization for distributed simulations of vehicular area networks. The proposed system is composed of two major

parts related to visualization and simulation; synchronization using messages over the communication layer, TCP/IP, is used for keeping the visualization consistent with the simulation outputs. A powerful gaming engine is also used to perform microscopic movements while the HLA framework is used for coordinating the simulation. Experimental results have shown that the synchronization between both parts generate overhead, observing that the control of vehicles by visualization components is a major bottleneck. As future work, improvements will be needed to enable the composition of simulations that consider more complex mobility models and a complete simulation of the communication between vehicles; such control might benefit from the use of agent-based simulations [36] for coordinating the movement of objects in the environment, and load balancing techniques [37], [38] will be employed, playing a significant role in the system. In the same line, components are expected to be added to interface with other simulation tools, such as SUMO and OMNeT++. Finally, the method used for generating the microscopic movement of vehicles will be modified to avoid the overhead on visualization components.

REFERENCES

[1] J. Härri, F. Filali, C. Bonnet, and M. Fiore, "Vanetmobisim: Generating realistic mobility patterns for vanets," in *Proc. of the International Workshop on Vehicular Ad Hoc Networks*. ACM, 2006, pp. 96–97.

[2] J. Harri, M. Fiore, F. Filali, and C. Bonnet. (2007) Vanetmobisim. [Online]. Available: http://vanet.eurecom.fr

[3] M. Fiore, J. Harri, F. Filali, and C. Bonnet, "Vehicular mobility simulation for vanets," in *Annual Simulation Symposium, 2007*, March 2007, pp. 301–309.

[4] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, 2012.

[5] D. Krajzewicz, J. Erdmann, L. Bieker, and M. Behrisch. (2014) Sumo: Simulator of urban mobility. [Online]. Available: http://sumo-sim.org/

[6] R. Barr, Z. J. Haas, and R. van Renesse, "Jist: Embedding simulation time into a virtual machine," in *EuroSim Congress on Modelling and Simulation*, 2004.

[7] R. Barr, R. van Renesse, and Z. J. Haas. (2005) Java in simulation time - scalable wireless ad hoc network simulator. [Online]. Available: http://jist.ece.cornell.edu

[8] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proc. of the International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ICST, Brussels, Belgium, Belgium, 2008, pp. 60:1–60:10.

[9] T. Steinbach and Kenfack, "An extension of the omnet++ inet framework for simulating real-time ethernet with high accuracy," in *SIMUTools 2011 – International OMNeT++ Workshop*. New York, USA: ACM DL, 2011, pp. 21–25.

[10] OMNet++ Community. (2014) Omnet++. [Online]. Available: http://www.omnetpp.org

[11] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[12] University of Southern California. (2014) The network simulator 2. [Online]. Available: http://www.isi.edu/nsnam/ns

[13] M. Piórkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, "Trans: Realistic joint traffic and network simulator for vanets," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, pp. 31–33, 2008.

[14] J.-P. Hubaux, J.-Y. Le-Boudec, M. Grossglauser, and P. Thiran. (2008) Traffic and network simulation environment. [Online]. Available: http://lca.epfl.ch/projects/trans

[15] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, "Traci: An interface for coupling road traffic and network simulators," in *Proc. of the Communications and Networking Simulation Symposium*. ACM, 2008, pp. 155–163.

[16] ——, "Traci: An interface for coupling road traffic and network simulators," in *Proc. of the Communications and Networking Simulation Symposium*. ACM, 2008, pp. 155–163.

[17] A. Wegener, M. Piorkowski, M. Raya, H. Hellbrck, S. Fischer, and J.-P. Hubaux. (2014) Traci. [Online]. Available: http://sumo-sim.org/wiki/TraCI

[18] C. Sommer, I. Dietrich, and F. Dressler, "Realistic Simulation of Network Protocols in VANET Scenarios," in *Conference on Computer Communications: IEEE Workshop on Mobile Networking for Vehicular Environments*. IEEE, 2007, pp. 139–143.

[19] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2011.

[20] ——. (2014) Vehicles in network simulation. [Online]. Available: http://veins.car2x.org/

[21] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," in *Proc. of the International Conference on Simulation Tools and Techniques*, 2009.

[22] F. Kaisser, C. Gransart, and M. Berbineau, "Simulations of vanet scenarios with opnet and sumo," in *Communication Technologies for Vehicles*, ser. Lecture Notes in Computer Science, 2012, vol. 7266, pp. 103–112.

[23] R. Fernandes, F. Vieira, and M. Ferreira, "Vns: An integrated framework for vehicular networks simulation," in *Vehicular Networking Conference*, Nov 2012, pp. 195–202.

[24] F. Ramm, J. Topf, and S. Chilton. (2014) Open street map. [Online]. Available: http://www.openstreetmap.org

[25] F. Ramm, S. Chilton, and J. Topf. (2010) Open street map: Using and enhancing the free map of the world. [Online]. Available: http://www.openstreetmap.info

[26] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *Pervasive Computing, IEEE*, vol. 7, no. 4, pp. 12–18, Oct 2008.

[27] Esri. (Accessed in 2014) Esri cityengine. [Online]. Available: http://www.esri.com/software/cityengine

[28] J. Halatsch, A. Kunze, and G. Schmitt, "Using shape grammars for master planning," in *Design Computing and Cognition*. Springer, 2008, pp. 655–673.

[29] B. Watson, P. Muller, P. Wonka, C. Sexton, O. Veryovka, and A. Fuller, "Procedural urban modeling in practice," *Computer Graphics and Applications*, vol. 28, no. 3, pp. 18–26, 2008.

[30] Autodesk. (Accessed in 2014) Autodesk 3ds max. [Online]. Available: http://usa.autodesk.com/3ds-max

[31] T. Boardman, *3Ds Max 8 Fundamentals*. Thousand Oaks, CA, USA: New Riders Publishing, 2006.

[32] R. H. Creighton, *Unity 3D Game Development by Example*. Packt Publishing Ltd, 2010.

[33] U. Technologies. (Accessed in 2014) Unity 3d. [Online]. Available: http://unity3d.com

[34] S. I. S. C. (SISC). (2010) Ieee standard for modeling and simulation (m&s) high level architecture (hla). [Online]. Available: http://standards.ieee.org/findstds/standard/1516.1-2010.html

[35] T. Pokorny and M. Fraser. (2007) The portico project. [Online]. Available: http://www.porticoproject.org

[36] M. Al-Zinati, F. Araujo, D. Kuiper, J. Valente, and R. Wenkstern, "Divas 4.0: A multi-agent based simulation framework," in *International Symposium on Distributed Simulation and Real Time Applications*, 2013, pp. 105–114.

[37] R. Alkharboush, R. De Grande, and A. Boukerche, "Load prediction in hla-based distributed simulation using holt's variants," in *International Symposium on Distributed Simulation and Real Time Applications*, 2013, pp. 161–168.

[38] R. De Grande, M. Almulla, and A. Boukerche, "Autonomous configuration scheme in a distributed load balancing system for hla-based simulations," in *International Symposium on Distributed Simulation and Real Time Applications*, 2013, pp. 169–176.