A Modular Distributed Simulation-based Architecture for Intelligent Transportation Systems

Robson Eduardo De Grande*, Azzedine Boukerche, Shichao Guan, and Noura Aljeri

PARADISE Research Laboratory - School of Electrical Engineering and Computer Science
University of Ottawa - Ottawa, Canada

SUMMARY

Simulations have been used extensively for evaluating scenarios which are very difficult, costly, or impractical to implement in real systems. Testing in a synthetic, realistic environment provides a means to determine the viability of solutions. Simulations have proved to be very useful in the verification of algorithms and protocols, offering tools for testing them in different situations. The simulation of vehicular area networks pose additional challenges as realistic mobility models are crucial and must be incorporated in scenario elements while applications and communication protocols are tested. Several simulators and simulation frameworks have been designed that aim to synthetically reproduce communication and mobility of vehicles as realistically as possible. The majority of such simulators merge pre-existing networking and mobility simulators, which add issues regarding compatibility and realism. Such simulators present limited run-time 3D visualization tools, essential for providing immersive environments. Therefore, in this paper we propose real-time simulation and 3D visualization for vehicular networks of realistic scenarios. This proposed simulation system generates output in real-time, making use of 3D-modeled real-world maps and effectively generating visualization as elements are updated in the simulation. Experiments have been conducted with simulation and visualization components to evaluate delays and performance of the proposed simulator. Copyright © 2015 John Wiley & Sons, Ltd.

KEY WORDS: Real-time Distributed Systems; Vehicular Networks; Intelligent Transportation Systems

1. INTRODUCTION

As scientific and technological advancements continue to facilitate the development of intelligent transportation systems and smart cities, Vehicular Area Networks (VANETs) have become crucial for enabling connectivity, solutions, and applications. Some solutions are effective in bringing entertainment and comfort to drivers, passengers, or pedestrians, but others are crucial building blocks for safety and environment-sensitive applications for enabling traffic management and control, public transportation, and accident prevention and mitigation. Some VANet solutions require the design of novel concepts and techniques in areas related to communication and information, possibly involving context awareness, service discovery, detection of objects, sparsity of the network, and associated applications. On the other hand, other solutions might also need to modify or improve the pre-established routing protocols, discovery and tracking mechanisms, and coverage and localization techniques as a means of empowering interoperability with pre-existing systems. A common characteristic for each VANET design according to its communication

*Correspondence to: School of Electrical Engineering and Computer Science - University of Ottawa - Ottawa, Canada. E-mail: rdgrande@site.uottawa.ca

Copyright © 2015 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls [Version: 2010/05/13 v3.00]
paradigms, which range over vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-sensor interactions.

Due to high demand, there has been an increase on the number of VANET-related projects and works; it is critical to undergo testing, evaluation, and validation before implementing any new method or solution in real environments, or to make them available to the public as services or integrated parts of systems. New solutions in VANETs must inevitably be evaluated to determined precision, efficiency, effectiveness, and practicability of proposed methods and mechanisms. Evaluations also provide the means of verifying the acceptance of a novel approach by the community and industry, and the development of standards that can ultimately be implemented in real-world scenarios. The nature of this class of systems in VANETs makes it impractical or excessively costly to conduct real experiments for most cases and scenarios, requiring the construction of prototypes or the synchronization of several elements on a medium- to large-scale. In order to evaluate a protocol or algorithm, some scenarios must be comprised of a large number of vehicles, matching the magnitude of real cities, while others are intrinsically required to analyze performance with the presence of real-world obstacles in urban and road environments; other scenarios impose long-term, extensive evaluations on particular analyses regarding performances or fault-tolerance. As a result, simulations are remarkably useful for extensively analyzing the behaviours of mechanisms and solutions in most diverse scenarios before they are considered ready to be implemented and deployed.

Building a simulator specifically for VANETs introduces several challenges, which are justified by the numerous simulation tools available to date. Each of these tools are built to achieve a specific objective, introducing limitations for scenario simulation. Some simulators present characteristics linking them to a specific VANET class of problems; other simulators partner with one another, aggregating features and functionalities from the simulation of networks and movement of objects. A large number of network simulators have been designed for verifying protocols, tackling them on the different layers of the networking stack, such as on MAC layer, routing layer, and transportation layer, as well as introducing cross-layer designs for more sophisticated techniques. On the other hand, many mobility and traffic control simulators have been developed in order to determine the behaviour of traffic in roads or urban areas, according to controlled applied management approaches. Since these two classes of simulators do not provide the necessary features for simulating VANETs, other simulation frameworks have been created that combine the networking simulations with highly mobile elements. Most of these frameworks basically add an interface between mobility and network simulators, generally losing precision and realism, depending on the methodology for connecting both parts. A common characteristic of all the frameworks is the lack of a real-time, realistic 3D visualization of the simulation execution; this visualization is necessary for inspecting microscopic details of particular simulation events.

In this paper, a system, previously described in [1], is proposed for coordinating simulations of vehicular area networks, modularly combining 3D visualization, real-time data updates, and third-party processing. This, system allows simulations to be integrated and enriched with external inputs of data and processing, as well as the output to an interface for users. Moreover, in this proposed simulation system, a realistic 3D visualization during the simulation run-time is introduced, noting that at this stage, realism is promoted through up-to-date, real maps. Together with the visualization interface, a sophisticated engine for gaming and physics is incorporated as a module in the system, further enhancing the realism of the simulations through microscopic vehicle movement. As the core element, the simulations are controlled through a distributed simulation framework. This coordination is performed under the circumstances of complex distributed systems [2], which leads elaborated interfacing to maintain such a symbiotic design [3] [4]. the development of a symbiotic with otherExperimental results show that simulations with a maximum number of objects are properly coordinated by the proposed system; design strategies are needed for handling large-scale simulations.

This paper is organized as follows. Section 2 describes the related work, providing highlights on challenging issues. Section 3 presents the proposed system by describing its architectural
components and their respective functioning. Finally, Section 4 succinctly summarizes the paper and presents directions for future related work.

2. RELATED WORK

Tools that evaluate vehicular network scenarios extensively and pragmatically are crucial for developing effective and efficient solutions. The need for these tools has stimulated the use of network simulators and mobility simulators but has essentially motivated the creation of several simulators that join mobility of road traffic and networking in wireless communications. The majority of practical simulators consist of the employment of a network simulator in which mobility models are incorporated to define the movement of objects. As a result, the tools listed and described in this section show two major components: (i) simulating the communication among vehicles and infrastructures and (ii) generating the mobility of objects on a determined map scenario, whose mobility output is statically or dynamically added to the network simulator.

The VANet scenarios are characterized by the high mobility of the elements, which then demands a simulator that can surely represent this mobility. Mobility simulators attempt to embed the behaviour of real vehicles through models in order to mimic the overall trend of real traffic.

VANET Mobility Simulation Environment (VANETMobiSim) [5, 6, 7] has been developed for the purpose of enabling a mobility simulation environment, which provides a means for the simulating mobility at the macroscopic and microscopic levels. Moreover, the simulations upon which VANETMobiSim are built are based on realistic automotive motion models. The simulation system is basically implemented in Java, and the maps are consolidated in the simulation environment through Geographical Data File (GDF) standard format. Realism is achieved through a set of several mobility, physics, and vehicular dynamics models. Mobility simulation output assumes the format of traces, which supports several tools and frameworks for mobile networks. The traces at the end contain a resulting mobility model in a static format. This restricts the network simulators to import trace files and build the entire simulation based on the predefined movement of objects. For just the purpose of visualization, auditing, and evaluation, some tools, such as HWGui †, help create a simple, restricted interface based on the output of VANETMobiSim.

Simulation of Urban Mobility (SUMO) [8, 9] is another mobility simulator, similar to VANETMobiSim. Largely developed by the Institute of Transportation Systems at the German Aerospace Center, SUMO is a road traffic simulator intended to provide the mobility simulation for large road networks. The simulator is available in a open source, portable code and aims to enable the simulation of vehicle movements at a very microscopic level. The simulator presents an architecture that enables portability, making itself extensible for adding elements specific to design applications. In addition to these characteristics, a main, valuable feature of this traffic simulator is its interoperability with other applications at run-time. This feature allows network simulators to update their simulations as objects dynamically move according to SUMO mobility models, providing more realistic simulation scenarios. However, this simulation tool presents limited simple visualization interfaces. These interfaces are essentially used for verifying the execution of urban mobility simulations. The interoperability and light-weight execution show promising designs in coupling SUMO with network simulators, which admit the dynamic or static input of data regarding the movement of objects.

Even though mobility is inherent in vehicular area networks, communication consists of of building up the core aspect from topologies that have formed over time. Thus, network simulators play a fundamental role for evaluating VANets. Several network simulators have been designed for either wired or wireless networks which consider the different topologies and technologies.

Scalable Wireless Ad hoc Network Simulator (SWANS) [10, 11] has been developed for simulating wireless and sensor networks. The simulator has been built using Java in Simulation Time (JIST), which is a java-based simulation platform offering a general-purpose discrete-event

†http://pi4.informatik.uni-mannheim.de/pi4.data/content/projects/hwgui/
simulation engine. The simulator has been developed with the objective of bringing efficiency and high-performance simulation executions. As a step further in the development, JIST/SWANS has been introduced specifically to simulate networking scenarios. Such scenarios present an emphasis on scalability, being composed of a large number of objects and imposing constraints on throughput. Experimental results of this simulator validate its efficiency, including several capabilities useful for network simulations; however, it concerns only aspects of networking, totally disregarding mobility models and requiring external tools for providing traces which represent the movements of the simulated objects in VANet schemes.

Similarly to SWANS, OMNeT++ [12, 13, 14] is also a discrete-event simulation framework designed specifically for conducting network simulations. Providing a very broad set of general-purpose tools, this framework allows for the simulation of wired and wireless networks in the most diverse cases, representing diversified IT systems, queueing networks, and hardware architectures. In order to assist and promote the design of simulations, the framework includes an object library, which introduces reflection, open data interfaces, reusability, modularity, and embedding support. Its modular architecture promotes access to different simulation features and allows the development of network simulations that can be easily extended and customized. This peculiar characteristic the dynamic modification of network topologies. Comparable to SWANS, OMNet++ does not present the simulation or emulation of mobility, so a third-party tool or simulator must be integrated in order to simulate VANets.

Another well-known system used for the simulation of VANets is the Network Simulator 2 (NS2) [15, 16]. NS2 is a discrete-event simulator that is completely focused on aspects of networking. This simulator has been extensively used for the verification and validation of tools, protocols, and solutions in the most diverse scenarios concerning networking research. The simulator is an outcome of joint efforts from many collaborators; it involves the simulation of wired and wireless networks, covering any topology or setup and supporting the full stack of communication, as well as the existing protocols, such as TCP/IP, UDP, and multicasting tool set. In order to evaluate the output of NS2 simulations, an animation tool named Nam ‡ is used. This simple visualization interface is based on Tcl/TK § ¶, and it makes use of static simulation traces to generate a limited, off-line visualization of the simulation execution. This simulator also requires traces of mobility models from other tools to generate the movement of objects in VANet scenarios.

Since a mobility simulator or a network simulator alone is insufficient for providing the minimum requirements to validate VANET protocols and algorithms, various solutions have been proposed. Thus, most of the current, and well-known, works consist of proposing a framework, or system, that combine both a mobility model and a network simulator. Most of these systems bring a static and limited simulator, based on static trace files that feed a network simulator.

Traffic and Network Simulation Environment (TraNS) [17, 18, 19] is an integration tool that has been developed to combine traffic and network simulations in one system. This integration takes place in SUMO and NS2. As a consequence, the interface between the two simulators is very limited due to the execution characteristics of NS2; more specifically, its constraints on interfacing with a mobility simulator require mobility trace files as input to indicate the coordination of movement of objects. This leads to simulations that contain unrealistic VANet scenarios, confined to predefined static paths. TraNS, additionally, attempts to decrease this lack of dynamism from NS2 by introducing a method that supports the on-line modifications on mobility in the traffic simulator through atomic commands from the network simulator; this method of interfacing allows run-time updates of the model. TraCI [20, 21] consists of the main part of such an interface, coupling the two simulators and allowing the movement of particular objects to be altered. However, this simulation environment presents a limited visualization interface, attached to NS and SUMO.

---

‡http://nsnam.isi.edu/nsnam/index.php/Nsnam:About
§http://www.tcl.tk/
¶http://wiki.tcl.tk/
Very similar to TraNS, Vehicles in Network Simulation (Veins) [22, 23, 24] introduces a simulation framework built specifically for VANETs, which concentrate entirely on vehicle-to-vehicle communication. However, this simulation framework integrates SUMO and OMNet++, two widely-used simulators for traffic mobility and networking, respectively. The two simulators are merged in Veins through bidirectional coupling; consequently, any update made on the objects, either in the road traffic simulator or the network simulator, is dynamically considered by both SUMO and OMNet++. Veins makes use of the TraCI interface to connect both simulators, which run in parallel. The visualization interface of Veins then inherits from SUMO and OMNet++ and presents the limiting characteristics, being restricted to a simple visualization for the purpose of verification and evaluation of implemented simulations.

Opportunistic Network Environment (The ONE) [25] introduces a system that provides simulations of both networking and mobility. This environment has been specifically developed for the evaluation opportunistic networks, so it brings a built-in emphasis on delay-tolerant networks (DTN). Since both networking and mobility are simulated together, the updates on the movement of objects are seamlessly shared in the environment. As an additional feature, The One allows mobility traces originating from third-party simulation tools, such as SUMO, to be incorporated into its simulation environment. Due to its focus on evaluating DTN routing protocols and algorithms, this simulation environment is restricted exclusively to DTN-related scenarios. This limitation aside, it offers a simple interface for real-time visualization.

Another simulation framework has been developed that functions equivalently to TraNS and Veins for providing VANet simulations [26]. This work joins OPNET ∥∗, a commercial network simulator, with SUMO, a mobility simulator. This simulator generates output files as a result of the simulation execution. The file is used by a graphical interface to visually reconstruct the simulation for verification.

Vehicular Networks Simulation (VNS) [27] is a framework that integrates a mobility simulator and a networking simulator. For this framework, the combination is made with DIVER T 2.0 [28], which is a road traffic simulator for large-scale environments, and network simulator 3 (NS-3) [29]. The framework aims to provide a complete integration of both parts towards a scalable simulation system. The simulator presents a visualization tool that can be used to verify the output of the simulations effectively. However, this visualization tool offers only a very limited 3D visualization interface, which might impose restrictions and lack realism.

According to the listing and description of major works for simulation of VANets, no simulation system can provide a real-time 3D visualization solution that can effectively work in parallel with ongoing simulations. A 3D visualization of simulations can greatly benefit the evaluation in runtime and allow expansion towards immersive environments that can be used for testing and training. The proposed simulator is expected to permit other systems, simulations or real environments to co-exist with ongoing VANet simulations. This co-existence is enabled through a modular architecture and a flexible simulator design. For introducing such a system, aspects that are directly related to guaranteeing real-time coordination and 3D presentation of simulated objects must be considered, such as challenges in synchronization, responsiveness, and data abstraction.

3. PROPOSED SIMULATION/VISUALIZATION SYSTEM

The proposed simulation-based distributed system introduces a modular architectural design for the purpose of allowing contiguous systems to cooperate with simulations. The initial version of this proposed system [1] described a simpler architecture in which enforced the visualization between simulation and visualization components. In this current step, the system focuses on the real-time 3D visualization of VANet simulations, so efforts are presented towards the use of tools for aiding the synchronization of parts and for manipulating 3D visualization. Third-party mechanisms are
employed in the development of the system for enabling 3D visualization, constructing virtual environments, and allowing interactions through a gaming perspective. All the integration of these tools and engines, as well as the use of different simulation infrastructures, is empowered through the flexible, modular architecture.

Even though the decoupled, modular design favours interoperability and facilitates adding other components in the system, it necessarily incurs synchronization among the communicating parts for properly coordinating the simulation execution and updating the visualization components. Most of the synchronization is generated to coordinate the movement of objects among the simulator, the physics engine, and the visualization tool and to initially match the positioning of objects in realistic maps used in the system. Since it is intended to introduce as much realism as possible, such maps are obtained from open sources and represent actual places and regions, needing proper synchronization to match them on the respective components of the simulation.

3.1. Architecture of the Proposed System

The system is basically divided in two layers partitioned in four groups, as depicted in Figure 1. In the outer layer, each group represents one part of an envisioned simulation system: Simulation, Visualization, Microscopic Processing, and Real-time Data. The inner layer of the architecture represents the integration interface, which is used to connect all the major parts of the system. This layer thus intermediates the communication in order to prevent data loss and guarantee the proper communication among parts. In this layer, the Communication Interface is the main component in receiving, buffering, and transmitting data to the endpoint; this design allows the endpoint to be located either in the same host or remotely. Parser is an essential component in the Communication Interface as it enables translation among the communication parts, particularly the communication between the simulator and the other joint parts; the parsing introduces a proper visualization and reciprocal communication.

Synchronization is enabled through mapping coordinates and commands in the simulation side of the system. This mapping is crucial for maintaining the consistency of object updates with the information kept in the simulation for the positioning of objects. Simulation objects and their respective versions in the other parts of the system must be consistently and correctly matched. Mapping is partially enabled through the Control/Update component in the simulation. Another component, known as Update Feedback Daemon, is employed to facilitate incoming updates from objects. This component works in parallel with the visualization or microscopic processing tools to store, buffer, and make data available from messages. This daemon is necessary for managing data without disrupting the execution of the upper layer components, 3D visualization tool, or the physics engine. Buffering of messages occurs by adding them up on a data structure based on queue, so incoming data is kept in order.

3.1.1. Simulation Components

In the simulation side of the system, the Map consists of an essential and basic component to provide a means for guiding the movement of objects during the execution of the simulation. The Map contains a set of coordinates which form a directional graph and the map information is stored in a data structure in the form of a linked list, containing a set of edges and points. A controller is introduced in order to facilitate access and filter the information inserted to the data structure. The controller verifies and manages the positioning of objects in each respective edge of the graph. In a first step, the consistency of the incoming positioning for the synchronization of objects is verified; after they are matched, the objects are assigned to their respective edge for fast and easy retrieval of their position according to the map structure.

Closely connected to the Map, Objects is a component that contains all the data that characterizes objects, as well as their current and short-term positioning in the simulation and visualization tool. A series of data structures is used to maintain the data of this component and to link its contents to the edges of the Map. The data structure is fundamental for quickly and effortlessly determining the status of an object. The component Objects also contain methods that are used to trigger updates on
stored object data; the updates consists of new calculations on the positioning of objects, which are later passed to the visualization. Such methods access the Mobility Model used in the simulation through calls to the Controller, and they work as an access interface the stored object data, triggering updates as soon as they determine, or predict, that the positioning is outdated.

The Mobility Model is ultimately necessary for determining a specific movement pattern for position updates. This component can contain a set of models that can be accessed to coordinate decision-making for the movement according to the currently used map. The model provides a decision for each object update. The simulation map provides the coordinates of the region delimited by the area surrounding the current position of an object; the region is composed by neighbouring edges in the graph. Through the region, the current status of surrounding objects is retrieved, allowing the next movement to be determined. The map also grants a global view to the Mobility Model, resulting in more sophisticated decision-making.

The Control/Update component is executed in parallel with the simulation. This component waits for, and responsively reacts to, update calls that might come from the Integration Interface; the update on the objects is performed by triggering the Objects component, which provides a new position of a given object. In addition to triggering requested updates, this component coordinates the manner in which the data is stored in order to maintain consistency throughout the simulation execution. The Control/Update works closely with the Data/Update Aggregator in order to organize the updated data that is sent to the other counterparts of the system. The Aggregator coordinates the communication between the simulation and other parts; it collects the messages that need to be transmitted and buffers them before transmitting over the network through the message passing communication. The Aggregator is intended to improve execution performance and aid synchronization by decreasing the frequency in which data is transmitted between parts.
3.1.2. Visualization Components

In the design of the proposed simulation/visualization system, the Visualization Components consist of a 3D Map, 3D Objects, and the full suite of third-party software represented by the Renderer. The Renderer provides all the necessary functionalities for dynamically rendering a given scenery that represents a virtual environment consisting of a map, objects, and structures. Although coordination of movement and status is performed for all objects during simulations, the rendering is realized only with the scenery that contains objects in the line of sight as a measure of improving performance by consuming resources in a useful manner. In order to enable data for smooth rendering, 3D Map and 3D Objects are introduced.

The 3D map is primarily responsible for storing and granting easy access to the map used for the visualization. This component allows quick reference for placing objects and contains a Map Controller. The Map Controller facilitates synchronization with the simulation, predominantly necessary for the startup phase of the simulation in order for maps to be fully matched, allowing a straightforward reference of positioning for later execution of the simulation. During the execution of the simulation, the controller is accessed at each incoming object update in order to verify the correct position of an object and perform adjustments as required, avoiding any inconsistency with the rendered scenery.

3D Objects concern all objects being manipulated by simulation and in the visualization, as well as the Object Controller. Each 3D Object contains all data related to an object, which essentially consists of 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 of the simulation.

3.1.3. Microscopic Processing Components

The unit concerning Microscopic Processing in the architecture of the proposed system is also comprised of a set of components. These components share some similarity with the Simulation Components, containing a 3D Map and 3D Objects. These two components present the same characteristics as previously described, and they assist the physics engine by providing necessary data for the proper calculations in determining the microscopic movement of objects over the selected simulation map. However, in this section of the architecture, the 3D Map and 3D Objects do not work passively as they do in the visualization; they send back information about the outcome generated from the microscopic processing, so that the rest of the components of the simulation/visualization can be updated accordingly.

The major component for microscopic processing is the physics engine, which basically defines and controls the microscopic movement of objects in a particular section of the simulation map. The processing provided by this third-party tool is highly necessary for providing calculations ruled by detailed physics transformations that are subject to and modifying the environment. Due to its granularity, it usually demands resources for the computations and directly influences the performance of the simulation and the visualization.

3.1.4. Real-time Data Components

Real-time data components are mainly intended for enriching the simulation with more accurate, realistic, and current data about the physical environment, represented in the simulation by the virtual 3D map. As these components provide data from specific elements that collect, filter, and aggregate information in an environment before sending to the simulator, three components are expected to feed such data: RSUs, Sensors, and the Transportation System. These components are intrinsically attached to or engaged in the real system, possibly an Intelligent Transportation System, so they are the elements best suited for feeding reliable information.

A significant function of these components is the parsing from the formatting used on them to the one employed in the simulator. This translation is necessary every time the data pertaining to traffic
conditions and events is collected and must be transmitted. Another important aspect to consider is the relevance, as well as the granularity, of the information collected to the simulation environment.

3.1.5. Integration Interface

The Integration Interface basically consists of the inner layer of the architecture presented in Figure 1. As the major middleware enabling this interface, the High Level Architecture (HLA) fundamentally provides a means for coordinating simulations through services and managing the communication between components, particularly through the Data Distributed Management (DDM) service. The DDM enables communication through subscription and publication, according to dynamically updated regions of interest. For instance, in another situation, Ownership Management service can be used to assign objects to be processed by the simulator or microscopic processing, or matched with real-time data.

HLA was designed as a general-purpose framework for distributed simulations, facilitating their development and coordinating their execution [30]. Indicated as a recommended process for creating interoperable parallel and distributed simulations, the framework also enables reusability, management mechanisms, and design principles. In order to maintain all these features, the framework consists of a set of rules, object model templates, and an interface specification. All these elements operate towards avoiding inconsistencies in HLA-based simulations, which are called federations, composed of interactive, independent simulations entities, known as federates.

The rules condition the federates’ behaviour, enforcing compliance with the specification. Object model templates define the manner in which federates communicate or interact through the exchange of information of objects. The interface specification delimits the methods federates must follow in order to access Run-Time Infrastructure (RTI) services and interact with other federates. The architecture of the HLA-based framework is simplified in Figure 2, showing that the RTI middleware is essentially composed of an RTI Executive (RTIExec) process, a Federation Executive (FedExec) process, and the RTI library (libRTI). The RTIExec controls the execution of federations and FedExecs; a FedExec manages federates and coordinates their life-cycle. The libRTI provides mechanisms as management services, which are needed to enable execution of the simulations. Thus, all communication between federates is coordinated and passes through the RTI, locally interfaced as the Local RTI (LRC).

A benefit of the coordination that HLA brings with its standards is the use of DDM, which introduces tools for minimizing communication overhead. The data exchange methods offered in the DDM, such as publish and subscribe, as well as interest management techniques, are major mechanisms for decreasing the amount of data transmitted during the execution of simulations.

3.2. Execution of the Simulation/Visualization System

The functioning of the proposed simulation system directly follows the organization of its major architectural components. Basically, the simulation runs in parallel with the other parts, assessing incoming data, updating the simulation accordingly, and sending data out to the concerned parts. Synchronization among all components, mostly between the simulation and visualization, is used
in order to keep the entire system working consistently. The synchronization guarantees that all simulated objects and their respective replications in other parts of the system show consistent status, particularly with the frequent updates that might occur during execution. Consequently, the coordination of information is enforced from the onset of a simulation and endures until its end, being realized through messages passing across the network or through shared memory.

In the initialization phase of the simulation, two major tasks must be performed to assure that all components are processing the correct scenario. These tasks are essential, and the remainder of the simulation only continues if they finish successfully. As described in Algorithms 1, 2, and 3, the initialization requires matching maps and the set of objects among the system parts; the matching is described by the similarity among the three algorithms, which must be executed in harmony to avoid inconsistencies.
The synchronization of the simulated map is summarized in a few steps, making use of pre-defined map reference points. As delineated in line 2 of Algorithms 1, 2, and 3, points are exchanged in order for the simulation part of the system to rotate, translate, and scale the map accordingly, as shown in lines 3, 4, and 5 of Algorithm 1. These actions conclude the synchronization of the simulated map, allowing the execution to continue to the next step, which consists in populating the environment with objects.

Basically, objects are generated in the Simulation Components, but they must match the elements that already exist in the respective real physical environment in case the simulation is being fed with real data, represented by line 3 of Algorithm 4. After objects are instantiated to represent vehicles and pedestrians, they are deployed on the map as presented in lines 7 and 8 of Algorithm 1.

The synchronization is triggered immediately afterwards, shown in line 9 of Algorithm 1, and the information is exchanged with the other parts, shown in line 3 of Algorithm 2 and line 3 of Algorithm 3. With this exchanged object information, 3D objects are created and deployed to the respective components, as shown in Algorithms 2 and 3. Together with the deployment, the initial mobility commands are determined by the simulation components, which are necessary for the microscopic processing.

The continuous execution, timely checking, and constant synchronization are represented by the while loops in Algorithms 1, 2, 3, and 4. All parts execute until an end signal is issued from the simulation, signalling the termination of the simulation, represented by line 23 of Algorithm 1.
the receipt of this termination signal, all other Algorithms (2, 3, and 4) interrupt their execution and shut down, as in their last line of code.

The simulation, more specifically the macroscopic simulation, runs based on the models it contains and on the incoming events originating from the microscopic processing or from real data. The visualization executes continuously, rendering the output previously generated by the simulation in run-time; this visualization constantly runs according to incoming updates as described in line 7 of Algorithm 2, refreshing the position of objects and rendering a specific view scenario on the simulated map.

Microscopic processing is triggered whenever detailed computations, dependent on precise physics aspects, are needed for a particular area of the simulation, or for the entire simulation. The updates generated by this processing are represented in line 7 of Algorithm 3; based on the incoming macroscopic decisions, the move method is called, verifying the incoming data for consistency against the current status and position of its local 3D objects according to their respective way points. The scripts behind these computations mimic the motion of real objects, making use of a physics engine. The way points are the major structural element allowing for the definition of short-range paths in a stretch of road segment in the simulation environment. Any object possessing movement ability presents a way point which it must follow the entire time. Each object thus contains a short list of way points it needs to chase, discarding a way point as it reaches it and picking the next way point on its list, as shown in line 8 in Algorithm 3. Contained in a long path, the list of way points is determined by the macroscopic computations of the simulator and might change during run-time.

There might be a case in which the simulation cannot provide frequent updates for the list of way points so that the list can present fewer elements. The Controller in the Microscopic Processing part is set to react preventively by requesting an update of the way points for a specific object, as described in lines 10-11 of Algorithm 3. The main objective in preventively requesting data is to ensure the microscopic movement of objects does not reach a state of suspension; the suspension can occur only if there is a long response delay of the simulation components. The same behaviour is observed for the visualization components, as shown in lines 9-10 of Algorithm 2. The list of way points is updated and increased, upon the receipt of a message from the simulation, as described in lines 14 and 15. In the event that there is a mismatch between the request current status of the simulated objects, the simulation controller needs to redeploy the objects according to a consistent position closest to where they were in the recent past, as shown in lines 13, 14, and 15 of Algorithm 1.

The simulation controller constantly updates movement position requests; the incoming requests are interpreted, verified, and responses are computed and sent back to the microscopic processing or visualization interface. This action is expressed in line 11 in Algorithm 1. The first required action upon the receipt of an object update request is to identify the correct edge on which an object might move, as described in line 12 in Algorithm 1. The find_edge method determines the closest edge to the provided position of the object, returning only one approximated edge. An inconsistency might occur, flagging a mismatch between the position of the 3D object and set of map edges; the mismatch may consist of an edge not being identified or an error of this computation might be larger than an accepted threshold, leading to the respective object to be redeployed, as previously explained. It is worth mentioning that when either redeploying or determining the movement of an object, the method move must consider conflicts over the entire set of current objects on the same edge; however, this verification is related to the used mobility model and its respective decision-making.

After the current edge of an object is identified, the next edge must be assigned to it, following a mobility model that determines through calculations the future positions that the object might present in a macroscopic scope, represented by line 17 in Algorithm 1. The mobility model is application-specific and considers several factors on its decision-making; in this particular design, the mobility model makes use of a on-demand scheme, allowing incoming real-time data from other sources that can influence the objects’ behaviour or that may need the adaptation of objects. The next edge fulfil the movement update request, or the current position and next edge related to a
A MODULAR DISTRIBUTED ARCHITECTURE FOR INTELLIGENT TRANSPORTATION SYSTEMS

(a) Initial 2D street grid  (b) Addition of layer-1 buildings
(c) Addition of layer-2 buildings  (d) Design of special landscape features
(e) Full 3D map of street grid  (f) Addition of textures on the 3D map

Figure 3. Process line of obtaining a full texturized 3D map

redployed object, is then translated to waypoints which contain the instruction on the future path an object needs to follow. This conversion is represented by line 19 in Algorithm 1.

The current design status of the proposed simulation and visualization system concerns only the movement of objects on a realistic map. Certainly, other elements are expected to be considered in the models to convey more precision and realism; such elements indirectly influence the objects’ movements. However, adding up these elements in the simulation show a potential increase in complexity of the simulation scenario, as well as in the entire system.

3.3. Tools and Implementation

A proof of concept for the proposed designed simulation/visualization system has been built through its implementation, with the use of third-party tools, which are incorporated into the simulation system through the HLA simulation framework. The tools are mostly linked to the components corresponding to visualization and microscopic processing, motivating the decoupled design of the system for better flexibility and adaptation.

The realism is a significant requirement for these simulations of vehicular networks, and the map composes the most important part of building up the simulation scenarios in such a context. Therefore, real maps are incorporated into the simulations from on-line sources, which are summarized in Open Street Map [31, 32, 33]. Open Street Map contains a database that is built based on local knowledge, which allows greater accuracy and current information in the maps since their updates are communally compelled. The retrieved maps can be represented in in ODbL format and exported into the standard Global Information System (GIS) data, OSM XML, for integration into the simulation system.

The obtained map is basically a graph composed by a set of edges and vertices. Consequently, this map needs to be populated by adding 3D elements that represent buildings and other objects that are necessary to credibly compose a landscape. In order to have these objects added, Esri City Engine [34, 35, 36], a third-party tool for 3D urban design, is used. The tool employs zoning for populating entire regions of a map and generates 3D city models from 2D GIS data. Autodesk 3DS Max [37, 38] can be used for modifying the 3D map structure, adjusting elements and adding up particular buildings where further work on modelling is required. These additional 3D objects may be reference points on the landscape of a specific scenario, such as towers, statues, or sophisticated buildings featured on the landscape of a real map.

Unity 3D [39, 40] is a powerful gaming design platform and has been used essentially for supporting the visualization and microscopic processing components in the proposed system. With the import of 3D generated realistic maps, it allows fast rendering and execution compatibility with
several platforms. Due to embedded support of many features around an industry-standard physics engine, optimization of performance, and notable characteristics in visualization and interface, the visualization and microscopic processing components are bound in this implementation to the Unity 3D design platform. Because of this particular design, 3D Map and 3D Objects components are shared between visualization and microscopic processing, aggregating a common controller to coordinate their actions during the execution.

Since HLA is being used as the main framework for the design and coordination of simulations, Portico RTI [41] is employed as the RTI middleware implementation to support the services that guarantee the communication intra-simulation among system components. This specific RTI is open source and multi-platform and is compliant with the HLA IEEE specification [30]. Thus, Portico RTI allows federates to be build, which directly contain the models of the simulations and work as the Integration Interface in the system.

### 3.4. 3D Realistic Maps

As mentioned in the previous sub-section, the map represents a major part of the simulation scenario in the scope of this work, so the support to realistic visualization is directly related to the realism of the maps. An initial effort has been expended to provide high realism representation of objects on the scenery maps, mimicking details of elements on the landscape.

The first step in generating the simulation maps is to obtain them from a source; in this work, they are derived from the online database of OpenStreetMap. As shown in Figure 3a, the map, which represents the city of Ottawa, Canada, is just a set of edges and vertices. Although functional for simulations, it totally lacks realism, requiring the following steps to process it.
The next step is comprised of adding 3D objects that represent real buildings. In an iterative fashion, objects are inserted like different layers, classified according to several zones in the map. The result of this step is presented in Figure 3d and 3e, in which suburbs are filled with houses, and the centre area composed of tall buildings. The result contains a landscape that resembles the real area as the map has been automatically populated through Esri City Engine. As an extension of this step, objects can be added to represent distinguished landscape elements. These elements are added to the map, making use of the direct 3D modelling of such elements through Autodesk 3DS Max. Following the example of city of Ottawa, Figures 3d and 3e respectively depict the modelling of the Canadian parliament buildings and a full map containing this prominent object.

In the final step, the process concludes by inserting textures to the map and 3D objects. The current simulation system concerns the controlled movement of objects around the map; thus, the simulation components contain a graph composed by the set of edges and vertices, and the visualization and microscopic processing components consist of the full textured map, as shown in Figure 3f. The synchronization among parts is comprised of data exchange and adjustment of the map’s features, so they properly match in both end points for the later execution.

4. EXPERIMENTAL RESULTS

Other experiments have been conducted in addition to the proof of concept for the proposed simulation/visualization system for vehicular networks. These experiments were intended to analyze the performance of the system and delays originating from the synchronization among the implemented parts, more specifically between the simulation components and the joint visualization and microscopic processing components. A simulation scenario was built in the proposed system with a map of the Ottawa downtown area, as depicted in Figure 3. The load of the simulation system was controlled through the number of vehicles deployed in the map, whose movements have been coordinated with a simple mobility model. The simulations were run on two computing servers interconnected through a gigabit Ethernet link; both servers presented an 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. For the performance analyses, three metrics have been considered: communication delay between the parts, processing overhead in the visualization components, and visualization performance of the final output of the system.

In the first set of analyses, the communication delays have been observed, and incoming and outgoing communication data, as well as overhead, are considered, from the simulation components of the system. With an increasing number of controlled vehicles (objects), the amount of bandwidth consumption is expected to grow proportionally. Figure 4 shows the incoming and outgoing packages in the computing server. There is a constant difference between the two curves (20 kilobits), which is explained by the fact that the incoming packages present a standard size larger...
than the outgoing packages. The bandwidth consumption grows in a linear manner with the increase of the number of objects in the simulation executions until it reaches the number of 130 objects. With scenarios containing more than 130 objects, the computation overhead becomes large enough to begin communication delays.

The other aspect important for observing the performance of the proposed simulation system is the processing overhead; it consists of the amount of time needed for the visualization components to update their 3D objects with the incoming data previously requested from the simulation. Figure 5 shows the results of the computation overhead, reflected through the response time of the simulation components and the update time of the visualization components. The visualization update times are divided in two categories, consisting of the updates in visualization with global view (Int. Update - GV) and focused view (Int. Update - FV). The difference between these two curves comprise a unique feature offered by the Unity 3D gaming engine: the focus that the visualization interface provides on processing and rendering the objects on a given scenery. Thus, the focused view means that only the elements in the line of sight of the gaming camera are processed while the global view means that the entire set of elements are processed. At the conclusion, the curves regarding the 3D object updates describe similar times; they show a slight increase in processing time as the number of objects for low values, but this increase ends with scenarios containing more than 50 objects. This behaviour is not observed in the simulation response time (Sim. Reply) since it decreases as the number of objects grows. Sim. Reply represents the entire time the simulation consumes to provide a response previously requested by the visualization components, comprised of the processing and communication times. Communication time plays a important role in this particular result since it contains the buffering time for sending the updated object data. Higher numbers of objects result in smaller waiting time for filling out the buffer. With scenarios composed of a few objects, there is a high delay in sending out updated data due to buffering time-outs. Figure 6a justifies this conclusion by showing the steady computational times for processing incoming object update requests, whose income frequencies increase as the number of objects increases; Figure 6b shows the cumulative processing times, which follows the increase in incoming load. Figures 8a and 8b show the average and cumulative processing times, respectively, for the incoming object update responses; even though the time consumed for computing a single object update is almost constant for an increase in the number of simulated objects, the cumulative computational time just follows the increase of load in a linear way. The drop in request/response frequency in both Figures 6 and 8 reflects the saturation on the microscopic processing for the movement of objects.

The third set of analyses focuses on specific observations of the visualization interface performance against the load imposed on the system, represented by the number of objects instantiated. In this particular analysis, performance is represented by the frames per second (FPS) of the output disclosed to a user. As presented in Figure 7, the analysis compares rendering performance using Global View and Focused View techniques. Since the larger number of objects and other elements are considered in the scenery, the global view is expected to require more
processing power. This is clearly shown in the attainment of lower FPS when compared to the focused view. For simulations involving a small number of objects (less than 50 objects), both views show the same rendering efficiency because of the light load. For situations with larger numbers of objects, but lower than 160, the focused view significantly improves performance; the global view presents a decrease in efficiency as the visualization load increases. However, a particular behaviour is observed in the results for visualization load higher than 160 objects: both curves tend to converge. These results are actually influenced by the performance of the microscopic processing; the number of objects is so high that the microscopic movement of vehicles in the 3D environment, which require detailed calculations regarding the physics engine, results in large overhead. In the end, the control of object movement consumes more resources than the rendering of the same set of objects. This overhead also influences communication performance from the simulation components since it increases the delay for the visualization components to request object updates in the synchronization process. The control introduces a bottleneck in the system, requiring more sophisticated mechanisms to assign the control of non-visualized parts of the environment to the simulation and relief for the microscopic processing components.

5. CONCLUSION

In this paper, a simulation and visualization system has been proposed for intelligent transportation systems. Such a system has been designed to enable the execution of realistic, real-time 3D distributed simulations of vehicular networks. Several challenges prevail in building the envisioned simulation and visualization tool, as one of the major obstacles is found in the synchronization among several parts of the system: simulation, visualization, microscopic processing, and real-time data. The synchronization is high, which is necessary to maintain consistency in the system, which mainly matches the visualization with the simulation output. For building the proposed system, several third-party tools have been used; these include HLA framework for controlling distributed simulations, as well as the communication among the parts in the system; physics engines for enabling microscopic processing; and a gaming engine for generating the visualization output.

To complement the proof of concept, experiments have been conducted in order to evaluate the performance of the devised system, whose results indicated that the synchronization still challenges the development of the system for larger-scale simulations. All the parts involved in experimental scenarios presented overhead and delays in communication and computation. However, the microscopic processing produced through the access to the physics engine was a major bottleneck in the entire system since it consumed a considerable amount of computation resources, generating wait times for the simulations and denigrating the rendering of the 3D environment, and consequently the final output to the end user. These results also drive our future work since further analyses are needed for improving performance through the use of alternative methods of
microscopic processing without negatively influencing the visualization of the simulation output in real-time. For enhancing the composition of simulations, more complex mobility models will be considered, as well as the use of agent-based simulations [42] for introducing more intelligent and realistic decision-making in the simulations or using other simulation tools, such as SUMO and OMNet++. Finally, as the scale of the distributed simulations increases, it is envisioned that the distributed simulation may have a stronger influence on performance, so more sophisticated methods for controlling the resources and load will be necessary, making use of load balancing techniques [43, 44]
A MODULAR DISTRIBUTED ARCHITECTURE FOR INTELLIGENT TRANSPORTATION SYSTEMS


