SCSimulator: An Open Source, Scalable Smart City Simulator*

Eduardo Felipe Zambom Santana[1,2], **Daniel Macêdo Bastista**[1], **Fabio Kon**[1], **Dejan S. Milojicic**[3]

[1]Department of Computer Science - University of São Paulo

[2]School of Engineering and Technology - Anhembi Morumbi University

[3]HP Laboratories - Palo Alto

{efzambom,batista,kon}@ime.usp.br,dejan.milojicic@hpe.com

Abstract. *Smart Cities, i.e., cities enhanced with a technological infrastructure that enables a more intelligent use and management of its resources, are currently seen as a powerful way of improving the quality of life of its citizens. Smart city platforms tailored at metropolises will be intrinsically very-large-scale systems; designing and developing such systems will be a daunting task. Also, deploying an infrastructure to test smart city systems is challenging due to costs, risks, and political difficulties. Therefore, being able to simulate the execution of smart city scenarios would be extremely beneficial for the advancement of the field. This paper presents a new open-source, large-scale smart city simulator capable of simulating different smart city scenarios and its performance and usability are also discussed.*

1. Introduction

Since 2009, most of the world population is living in cities [United Nations 2009] and current resources and infrastructure are hardly enough to cope with the increasing demand generated by population growth and geographic concentration [Caragliu et al. 2011]. Thus, making cities smarter can help optimize resource and infrastructure utilization in a more sustainable way. However, deploying the infrastructure to test Smart City systems will be a considerable problem due to costs, risks, and political difficulties.

There are already a few Smart City experimental testbeds such as SmartSantander [Jose et al. 2013], with 20 thousand sensors, and Padova Smart City [Zanella et al. 2014], with 3 thousand sensors. However, a Smart City has more components that are complicated to deploy in a testbed such as Smart Buildings, Smart Grids, and Vehicular Ah-Doc Networks (VANET). Another problem is that the current testbeds are deployed in small to medium cities. Deploying a sensor network in a huge city such as São Paulo, with 11 million inhabitants, is a daunting task.

An alternative to support large-scale tests and experiments is the use of simulators. The very large amount of actors in a Smart City environment (e.g., consider the 8 million cars and 11 million people in the city of São Paulo) requires a very-large-scale

*SCSimulator code, videos, and documentation are available in its page: https://github.com/ezambomsantana/smart-city-simulator

simulator. To be able to simulate such huge scenarios, we are developing SCSimulator to support tests of many Smart City complex scenarios such as traffic, energy, and disaster management. SCSimulator will certainly be very useful for many stakeholders such as city administrators, smart city software developers, and operators of city systems.

This paper is organized as follows. Section 2 presents related work. Section 3 describes the simulator functional and non-functional requirements. Section 4 presents the architecture and implementation of SCSimulator. Section 5 presents the SCSimulator performance and usability evaluation. Finally, Section 6 points out our conclusions and future work.

2. Related Work

In our literature and Web searches for Smart City simulators, we did not find any simulator that is capable of simulating large-scale and complex scenarios with multiple actors such as cars, buildings, people, and sensors. We found some specific simulators tailored at specific domains such as Smart Grids, VANETs, and car traffic, which we describe below.

DEUS (Discrete-Event Universal Simulator) is a discrete-event general purpose simulator, which was extended to simulate a Vehicular Ad-Hoc Network (VANET) [Picone et al. 2012]. In this Java-based, open-source simulator, it is possible to extend the base Node and Event model to implement specific actors to simulate entities such as cars, buildings, people, and sensors. Due to its architecture and non-parallel Java implementation, its scalability is weak, which was confirmed by experiments with almost 10 thousand nodes that we carried out.

Veins is a VANET simulator [Darus and Bakar 2013] integrated with OMNET++[1], a well-known discrete-event network simulator, and SUMO (Simulation of Urban Mobility)[2], a traffic simulator. In Veins, it is possible to simulate traffic scenarios such as accidents and traffic jams. In our experience with it, it was difficult for us to understand its code and architecture and running it in parallel mode was not trivial.

Siafu is a Java agent-based, open-source simulator [Europe 2007] used to simulate mobile events in a city. The simulator has a user interface to visualize simulation data and can export data sets. In Siafu, the agent creation is manual, so it is more appropriate for small, simple scenarios that can be visualized via a simple graphical interface.

3. Requirements

In this section, we present the functional and non-functional requirements that a Smart City simulator must handle.

3.1. Functional Requirements

To find the functional requirements for the initial version of our SCSimulator, we reviewed the literature on smart cities domains [Jose et al. 2013, Zanella et al. 2014]. We then selected the three most commonly explored domains in the scientific literature, which include the scenarios and actors described in the following.

[1]OMNET++ - https://omnetpp.org
[2]SUMO - http://sumo.dlr.de

- **Traffic:** It should be possible to simulate multiple traffic scenarios such as the impact of improving the quality of public transportation, the cause and effects of traffic jams, and the best routes for public transport systems. The main actors of these scenarios are vehicles, people, traffic lights, and flow sensors.
- **Resource Usage and Distribution:** In a Smart City, it is possible to manage the use of city resources such as water and electricity. It is possible to measure the amount of these resources at many levels such as buildings, streets, and neighborhoods. It should be possible to simulate the amount of water and electricity used by buildings as well as how these resources are distributed across the city. The main actors of these scenarios are Smart Buildings, Sensors, and Energy Stations.
- **Waste Management:** Many authors cite the management of waste services in the city as an important Smart City domain. It should be possible to simulate the use of a sensor network to notify when trash cans are full, visualization and control of trash vehicles, and prediction of when waste disposals will be full with the growth of population. The main actors of these scenarios are buildings (generate and receive trash), vehicles (trash truck), and sensors.

3.2. Non-Functional Requirements

The most important non-functional requirements necessary to implement a Smart City simulator are (1) enabling ease of use of the simulator and (2) the execution of very large simulations. Thus, the main non-functional requirements are:

- **Scalability:** To simulate Smart City scenarios, it will be necessary to manage millions of actors such as cars, people, buildings, and sensors. Therefore, the simulator scenarios have to scale from hundreds to millions of actors. To achieve this, distributed and/or parallel simulations will be mandatory.
- **Usability:** The simulator has to allow easy description of the simulated scenarios, enabling people that do not know the internal implementation of the simulator to develop scenarios with little effort. Thus, the programming model has to be intuitive and independent of the internal implementation of the simulator.
- **Extensibility:** It is unlikely that a simulator will provide all required features for Smart City simulations. The simulator has to be easily extensible, offering simple mechanisms for implementing new actors and changing their behavior, for implementing new metrics as well changing the behavior of the simulator itself. So, it is important not only that the simulator be open source but also that it is well documented and is implemented with good quality, extensible code.

4. SCSimulator

The goal of SCSimulator is to meet all the requirements stated in the previous section. To achieve that, our approach has been to reuse as much good quality code as we can. Thus, we based the project on an open-source, large-scale, discrete-event simulator called Sim-Diasca (Simulation of Discrete Systems of All Scales) [Song et al. 2011] developed in France by the EDF energy company.

Sim-Diasca is a general purpose simulator that has the goal of enabling very large-scale simulations. This simulator is implemented in Erlang, a functional language that facilitates the implementation of massively parallel and distributed applications. Moreover,

Sim-Diasca has a simple programming model enabling fast development of simulation scenarios. Our experiments with Sim-Diasca demonstrated that it scales much better and is much easier to use and extend that the other simulators mentioned in Section 2

Figure 1 presents the simulator architecture. The bottom layer is the Sim-Diasca simulator, responsible for the discrete-event activities such as Time Management, Random Number Generation, Deployment Management, and the Base Actor Models. The middle layer is the Smart-City Model we developed as part of our research, which implements the required actors for Smart City simulations such as vehicles, people, and sensors. The top layer comprises the scenarios that are implemented using the Smart City model.

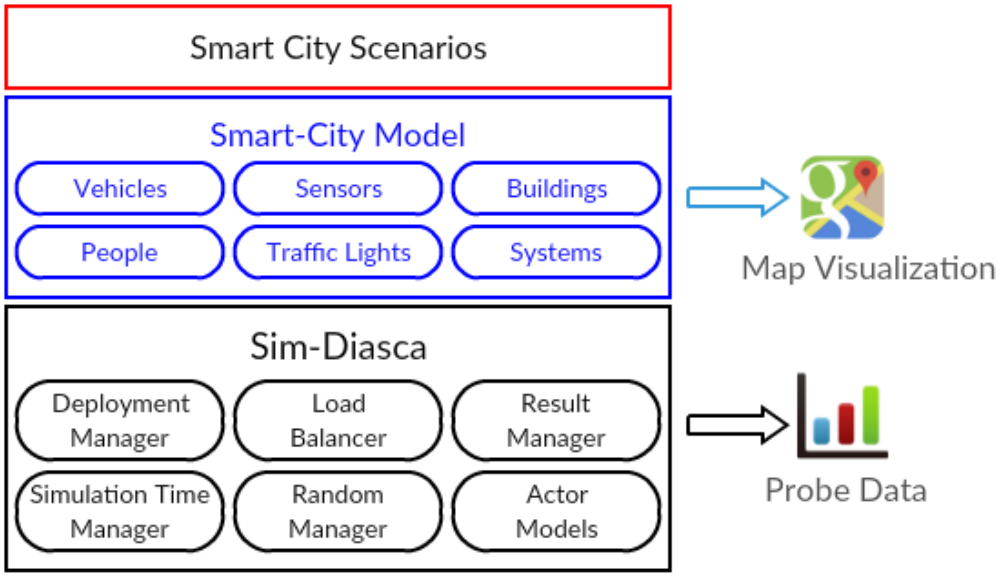

Figure 1. SCSimulator Architecture

With the SCSimulator, there are two ways to visualize the results of the simulation. First, it is possible to associate Probes with each actor and, anytime in the simulation, the actor sends data about its state to its probe. At the end of the simulation, a file with all the values passed to a probe is created. The simulator can also generate graphs with probe values over simulation virtual time. Figure 2 shows an example of a chart with the values of the probe of a sensor actor.

The second way to visualize the results of the simulation is via an animated GUI based on the GoogleMaps API. In this case, it is possible to visualize data gathered from the simulation while the simulation is running. Figure 3 shows an example of the simulation map, showing a simple simulation with a few dozen cars, buses, sensors, and traffic lights.

5. Simulator Evaluation

In this section, We compared SCSimulator with the three other simulators presented in Section 2. We compared three aspects of the simulators: Scalability, Programming Model, and Output.

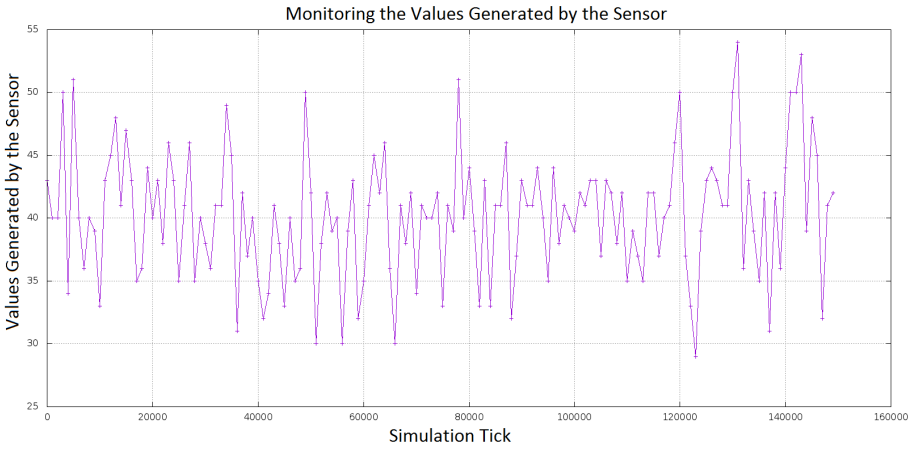

Figure 2. Probe generated data

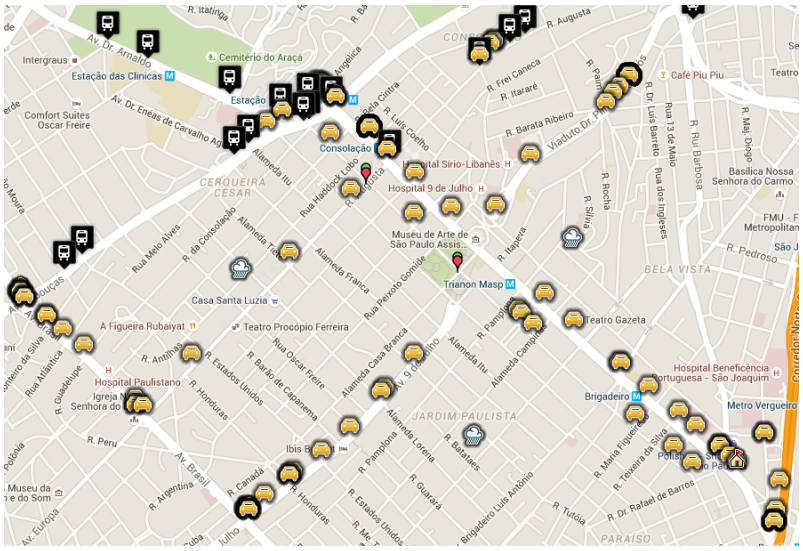

Figure 3. Animated Visualization with the SCSimulator

5.1. Scalability

To evaluate the scalability, we developed a scenario with some actors in SCSimulator. The initial configuration of the scenario had four buildings, four sensors, and three bus terminals. Cars came out of the buildings at a certain rate and buses left the terminals at another rate. We executed the simulation twice, the first with 500,000 simulated clock ticks and the second with 10 million simulated clock ticks. The first simulation took approximately 2,500 seconds and the second 100,000 seconds. To run the simulations, we used a machine with AMD FX6300 processor with 6 cores and 10 GB of memory running the GNU/Linux (Fedora 21) operating system.

Figure 4 shows the number of actors created during both simulations. The first simulation started with 6 actors and finished with 896. The second simulation also started with 6 actors and finished with 18128. The figure shows that the number of actors grew almost linearly with wall-clock time, indicating that the simulator could handle the load easily up to 18,000 actors and 10 million simulated clock ticks.

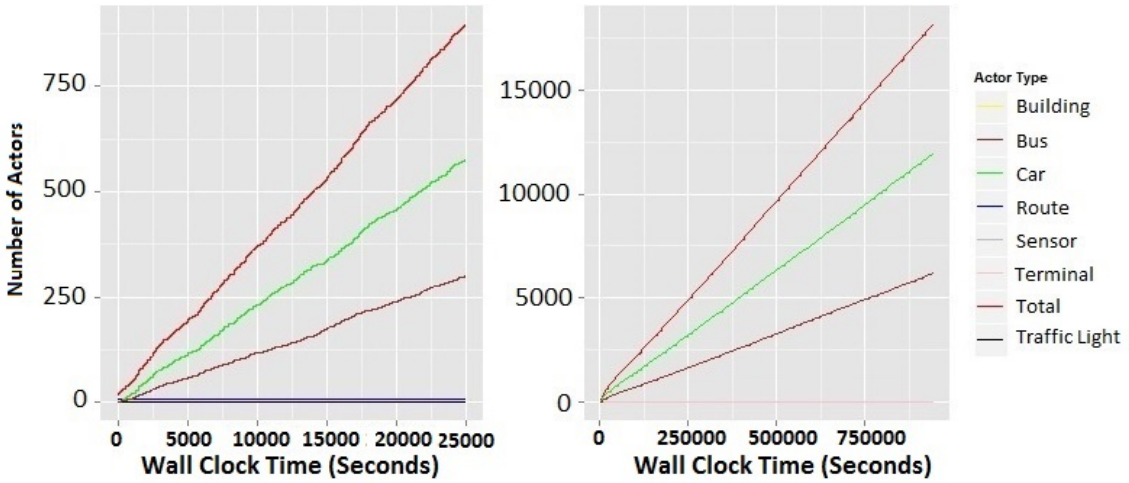

Figure 4. Number of actors over time

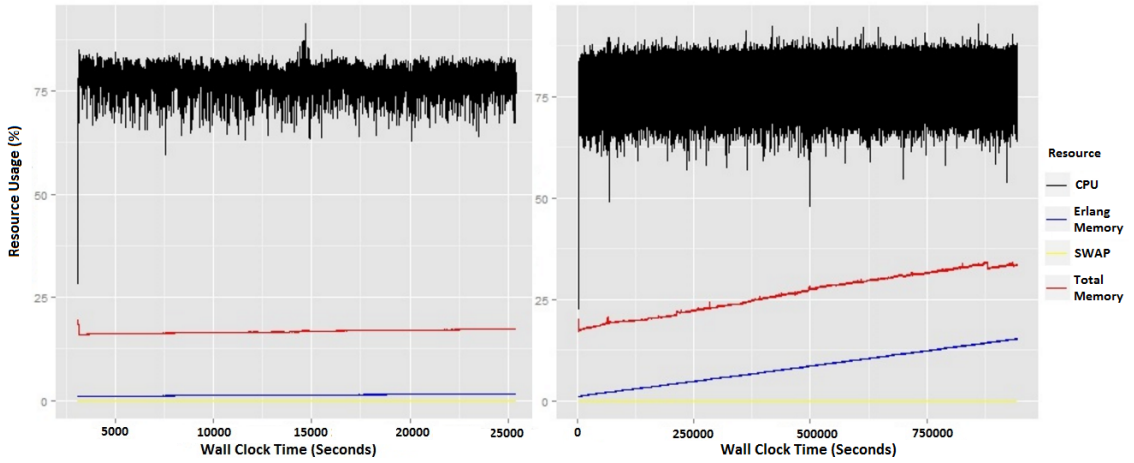

Figure 5. Computational resources used in the simulation over time

Figure 5 shows the resource consumption of both simulations. In the first simulation, the results show that resource usage almost did not change across the simulation. With 6 actors, the Erlang VM used 0.8% of the machine memory; with almost 900 actors, the amount of memory used increased to 2%. CPU usage also did not change very much over the simulation time; the chart shows that the average of the usage was almost the same during all the simulation. In the second simulation, the amount of memory occupied by the Erlang process also started in 0.6% and finished in 15.6%. The amount of memory used by the other processes of the machine increased almost at the same rate of the Erlang VM in both simulations. This occurred because creating an Erlang thread needs an OS action, and some processes create a file to write the values of the simulation.

These results show that the CPU is the bottleneck for the simulations up to the number of actors we experimented with, pointing to the necessity of parallelizing the execution of the simulation in more cores and/or machines.

We developed a similar simulation in DEUS and Siafu and executed on the same machine. DEUS stopped working with 7320 actors because of an out of memory error. In

Siafu we created a simulation with 384 actors but, as said in Section 2 it is hard to create a large scenario. These two simulators only execute sequentially. We did not present results obtained with Veins simulator because the simulator does not export the current number of actors in the simulation, but we could not create a very large simulation.

5.2. Programming Model

The programming model is important to evaluate the extensibility and usability of the simulator. The SCSimulator and DEUS have a very similar programming model. Both provide a base class (Actor in Sim-Diasca and Node in DEUS) that developers can extend to implement the simulation actors. The Siafu programming model is a little different, and the programmer has to understand all the code of the simulator. To create a traffic simulation in Veins no code is required, but if it is necessary to add new components to the simulator, it is necessary to change OMNET++ and SUMO and its communication. Therefore, SCSimulator and DEUS seem to be more easily extensible than Siafu and Veins.

To verify usability, it is important to analyze how to create the Smart City scenarios. In SCSimulator, an Erlang code describing the initial configuration of the simulation is required to define a scenario. Listing 1 presents an example of how to define the scenario actors. In the listing, a traffic light and a sensor actor are defined.

```
class_Actor:create_initial_actor( class_TrafficLight,
    [
        _TFNAME="traffic_light_1",
        _TFLAT=-23.562831,
        _TFLONG=-46.656866,
        _TFTIME=10
    ]),

class_Actor:create_initial_actor( class_Sensor,
    [
        _SName1="sensor_1",
        _SLat1=-23.570813,
        _SLong1=-46.656108,
        _SType1 = "temperature"
    ]),
```

Listing 1. SCSimulator Actors Definition

Veins and DEUS have a similar way of defining the scenario using XML files that describe the initial actors and their corresponding behavior what leads to an additional overhead for parsing. Siafu has a visual interface to define the scenarios, which is good and easy to define small simulations, but which makes impractical the creation of large simulations with a large number of actors.

5.3. Simulation Output

The SCSimulator has the two outputs already presented: the animated visualization based on Google Maps and the Probe datasets and graphs generated by the simulator. DEUS also uses Google Maps to present the results of the simulations but does not generate values to be analyzed after the simulation. Veins and Siafu have their own interfaces to show the simulation results; Siafu also generates datasets with values generated in the simulation for post-mortem analysis.

6. Conclusions and Future Work

This paper described the development of SCSimulator, a simulator that aims to advance the state of the art in the integrated simulation of Smart Cities, offering scalability and an easy programming model. In this first version of the simulator, we implemented actors of a Smart City environment such as Smart Buildings, Vehicles, and Sensors. The experiments showed that the simulator is scalable, a fundamental requirement to simulate Smart Cities. Compared to other simulators, SCSimulator is also easy to use and the results of the simulations can be obtained both by lists of values, graphs or an animated simulation with a GUI

In our ongoing work, we are experimenting with larger scenarios, going up to hundreds of thousands of actors; for that, we need to execute the simulator in larger machines, with more cores so as to better explore the parallelism supported by the Actor model of the Erlang language. In the long run, we intend to perform simulations with millions of actors but we anticipate that several challenges and bottlenecks will need to be resolved before we can achieve that. As future work, we intend to implement other Smart City scenarios such as disaster management and public transportation. Finally, we intend to perform a functional evaluation with city officials and public policy makers to validate the simulated scenarios and improve the simulator usability.

7. Acknowledgments

This work is funded by Hewlett-Packard Brazil.

References

[Caragliu et al. 2011] Caragliu, A., Del Bo, C., and Nijkamp, P. (2011). Smart cities in europe. *Journal of urban technology*, 18(2):65–82.

[Darus and Bakar 2013] Darus, M. Y. and Bakar, K. A. (2013). Congestion control algorithm in vanets. *World Applied Sciences Journal*, 21(7):1057–1061.

[Europe 2007] Europe, N. (2007). Siafu: An open source context simulator.

[Jose et al. 2013] Jose, A. G., Gutiérrez, V., Santana, J. R., Sánchez, L., Sotres, P., Casanueva, J., and Muñoz, L. (2013). Smartsantander: A joint service provision facility and experimentation-oriented testbed, within a smart city environment.

[Picone et al. 2012] Picone, M., Amoretti, M., and Zanichelli, F. (2012). Simulating smart cities with deus. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, pages 172–177. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Song et al. 2011] Song, T., Kaleshi, D., Zhou, R., Boudeville, O., Ma, J.-X., Pelletier, A., and Haddadi, I. (2011). Performance evaluation of integrated smart energy solutions through large-scale simulations. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 37–42.

[United Nations 2009] United Nations (2009). Urban and rural areas 2009.

[Zanella et al. 2014] Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *Internet of Things Journal, IEEE*, 1(1):22–32.