Simulation of Cell Signaling Communications Using Event-Driven Algorithms

Daniel Huertas Gonz áez and Alfonso Rojas Espinosa

***Abstract*—In this paper we present two implementations of event-driven algorithms for simulating molecular dynamics using the Omnet++ Simulation Framework and its Future Event Set (FES) implementation. The first one uses a cell-linked list algorithm. The second one extends the cell-linked list algorithm incorporating a Verlet neighbor list algorithm. We also present results and compare both algorithms over a set of different scenarios. Finally, we discuss the advantages of using the Omnet++ Simulation Framework and the implemented algorithm for simulating cell-signaling communications.**

***Index Terms*—Event-driven, molecular dynamics, Omnet++, particle system.**

I. Introduction

Cells, in terms of biological organisms, are able to perceive and produce a response to their environment accordingly. For unicellular organisms, the ability to react to changes in their environment is key for survival, whereas for multicellular organisms the exchange of information between neighbor cells governs basic cellular activities and actions. This communication between cells is known as cell signaling. There are different types of cell signaling communications: cells can communicate with each other via direct contact (juxtacrine signaling), over short distances (paracrine signaling) or over large distances (endocrine signaling).

In this work we focus on the subset of juxtacrine and paracrine signaling types, where event driven simulation algorithms can be easily applied. Specifically, we will focus on the simulation of large systems of particles modeling the communication channel using a hard-sphere system approach. Furthermore, since our goal is to provide the tools to study these types of communications, we present an implementation of two event-driven algorithms implemented using the Omnet++ [1] framework simulator. We also comment some of the results of running the algorithms on a set of different scenarios.

To conclude, we discuss the advantages of using event-driven algorithms on cell-signaling communications, but we also point out the difficulties to model realistic scenarios.

II. Event-Driven Molecular Dynamics

In molecular dynamics there are two different, well-known simulation methods [2]. The first one, known as time-driven simulation, divides the simulation in small steps and performs the required computations after each time step. The other one, referred to as event-driven simulation [3], [4], defines a series of events that take place during the simulation process, which are then processed one after the other. The main difference from the time-driven method is that the simulation time does not advance in small steps, but it advances the difference in time between events, thus jumping in time from one event to the next. Time-driven algorithms tend to be inaccurate when the time step is large (in terms of processed events), and become more accurate as the time step tends to zero. On the other hand, event-driven algorithms tend to keep accuracy since automatically adjust the time step.

One example of molecular dynamics using a time-driven simulation method applied to molecular communications is the N3Sim [5] from the N3Cat Initiative, which has already brought valuable results on the field of molecular communications [6].

Although both simulation methods were considered we chose to implement an event-driven algorithm [7]. The main reason to do so is that it best suits for a network simulation framework like Omnet++, since it already implements a Future Event Set (FES) that can be easily adapted as a self-ordering event heap for the algorithm. The main structure of the algorithm is based on the hard-spheres model, although it can be easily extended to other type of models such as fluid and particle interaction models.

The hard-spheres model, also referred as the billiards model, consists of a collection of non-overlapping spheres (or disks in a 2D model) contained within a bounded region, each moving with a certain velocity. The main features of the model are that the spheres follow the Newtonian laws of physics, that is, particles move along simple, deterministic paths in between collisions. Also binary collisions are considered to have no duration and involve deterministic changes of velocities of the colliding particles. Furthermore, elastic collisions are considered, thus conserving the total momentum and kinetic energy of the system.

$$m_1\vec{u}_1 + m_2\vec{u}_2 = m_1\vec{v}_1 + m_2\vec{v}_2 \quad (1)$$

$$\frac{1}{2}m_1u_1^2 + \frac{1}{2}m_2u_2^2 = \frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 \quad (2)$$

Secondly, different types of events must be considered: sphere to sphere collision evens and sphere to boundary collision events are the main ones, but other events must be considered to help reduce computational costs.

Finally, we model the cells, the signaling molecules and

Manuscript received November 10, 2013; revised January 15, 2014.

D. Huertas is with the Polytechnic University of Catalonia, Barcelona, Spain (e-mail: huertas.dani@gmail.com).

A. Rojas is with the Telematics Engineering (ENTEL) Department, Barcelona Telecommunication Engineering Technical School (ETSETB), Polytechnic University of Catalonia (UPC), Barcelona, Spain (e-mail: alfonso@entel.upc.edu).

DOI: 10.7763/IJCTE.2014.V6.880

the medium particles by choosing different sphere radius, mass and velocity values accordingly.

III. The Algorithm

A. The Main Loop

Starting from a naïve approach, the algorithm is composed of the following steps:

S1: Compute the time of the next collision event.

S2: Advance all the particles in the system up to the computed time.

S3: Change the state of the particles associated to the computed event.

Then repeat these three steps up to the required simulation time. This oversimplified algorithm can be applied as long as the number of particles in the system is not very large. But this algorithm does not scale well with the total number of particles since the required number of calculations that have to be performed each iteration depends on the number of particle pairs as follows

$$N_c = \frac{1}{2} n(n-1) \tag{3}$$

Then it is clear that some changes need to be made on the previous algorithm to improve the performance and scalability [8]. The first one is to save collision times, since on two consecutive iterations the computed time on the first iteration will likely be the same on the second one. Is at this point where we make use of the Omnet++ Future Event Set (FES) implementation, and saving the collision times in its time ordered queue of events. The second change comes from the realization that a collision between two particles is not likely to affect distant particles in the near future. So in order to reduce the number of pairs that need to be checked for each particle we implement a cell list algorithm.

B. Cell List Algorithm

As Fig. 1 shows, a cell list algorithm consists in dividing the simulation space into smaller cells and link the particles to the region they are in. Then, when a particle needs to compute the next collision event it only needs to check the pairs with particles in cells nearby, since distant particles are not likely to interfere.

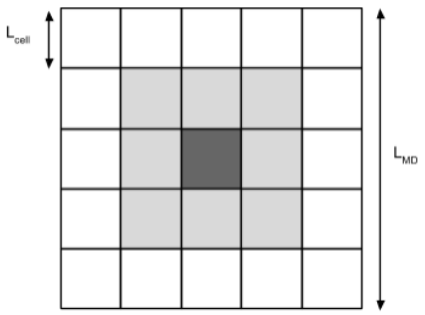

Fig. 1. Simulation space divided into smaller cells.

This introduces a new type of event, conveniently called a transfer event that needs to be handled in the algorithm. This event is produced when particles leave one cell and enter another one, and ensures that no collisions are overlooked. This may seem that adds extra computation to our algorithm. However, since distant particles are not constantly checked for collisions (which is an expensive computation in terms of CPU cycles) the overall performance is improved. Furthermore, these transfer events will also be saved in the Omnet++ queue of events to save the algorithm to be constantly checking for them.

Nevertheless, since we need to create a list of particles for each cell, this algorithm has a higher cost in memory usage: the smaller the cell size the higher the number of lists needed and the transfer events that need to be handled. We can see that there is a lower limit for the size of the cell and thus in the total number of cells: a cell can be no smaller than the diameter of a particle. Also, the finer the grid, the fewer pairs that need to be checked.

At this point, our algorithm is formed of the following steps:

S1: Compute the time of the next event, be it a collision event or a transfer event.

S2: Advance all the particles in the system up to the computed time.

S3: Handle the event, that is, change the particle states in case of a collision event, or update the cell lists in the event of a transfer.

These three steps are conveniently placed in the event handler that every Omnet++ module has to implement. This is a relevant difference with previous hard-sphere event-driven algorithms seen so far. Extending a sphere as an Omnet++ module and placing the steps from the main loop in its event handler we can remove step 2, which leaves the algorithm with two main steps:

S1: Compute the time of the next event, be it a collision or a transfer.

S2: Handle the event, that is, change the particle states in case of a collision event, or update the cell list in the event of a transfer.

This change gives the algorithm an asynchronous approach, meaning that not all the particles are at the same simulation time. This is easily handled by saving the last event time for each of the particles.

C. Verlet Neighbor List Algorithm

In order to further improve the performance of the algorithm, we have also included a Verlet Neighbor List algorithm, or also known as Near-Neighbor List algorithm [9]. This algorithm consist in skipping those particles that are further away from a given cut off radius (R_{cut}), only taking into account the ones that fall inside, creating the so called neighbor list. As Fig. 2 illustrates, only the particles that are closer than R_{cut} are taken into account. Collisions are then checked only with those particles that are in the neighbor list. In our case, the neighbor list algorithm uses the cell lists to retrieve only the particles in the neighboring cells. Then, applying the cut off radius further crops the list of particles to be checked. However, this list will change over time and needs to be updated. We then define the out-of-neighbor event, that is, the current particle has left the neighbor area and needs to update its neighbor list. Furthermore, this list update is also performed when the particle leaves the space cell where it is listed.

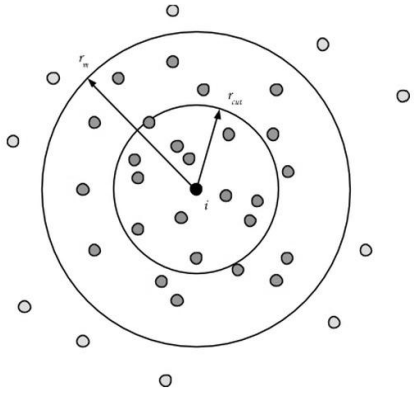

Fig. 2. 2D particles falling inside and outside of the cut off radius of the i-th particle.

D. Events Translation and Omnet++ Visualization

From the previous algorithms we must translate three types of events into Omnet++ messages. Therefore, we define three different kinds of Omnet++ messages:

1) Collision message, whether it is a particle to particle or a particle to boundary collision.
2) Transfer message.
3) Out-of-neighbor message.

These messages are then managed by Omnet++ and delivered to each particle (module). Every time a message is delivered it will trigger the execution of the steps before mentioned, computing the next event and updating the simulation state each time, until the desired simulation time is reached.

At the same time, a novel web server module has been introduced into the simulation algorithm. Omnet++ comes with tcl/tk visualization support, but with no 3D space representation. The idea behind this custom, self-content module is to surpass some of these limitations and offer the ability to visualize the current simulation status. This web server runs on an independent, parallel thread and offers real time 3D visualization using any of the latest web browsers using the WebGL technology [10]. This opens the door to a whole new set of tools to manage molecule dynamics simulations with Omnet++.

IV. Result

In this section we validate that the algorithm behaves as expected through a series of experiments. First we use different initial particle distributions to compare how the algorithm behaves. Secondly, we compare the running time of both algorithms over a set of different configurations parameters. Finally, we suggest possible modifications to better fit the algorithm for the simulation of cell signaling communications. All the experiments are performed using a 3D simulation space. The results presented hereafter have been obtained running the algorithm on a x86_64 Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz GenuineIntel GNU/Linux, 8.0 GB RAM.

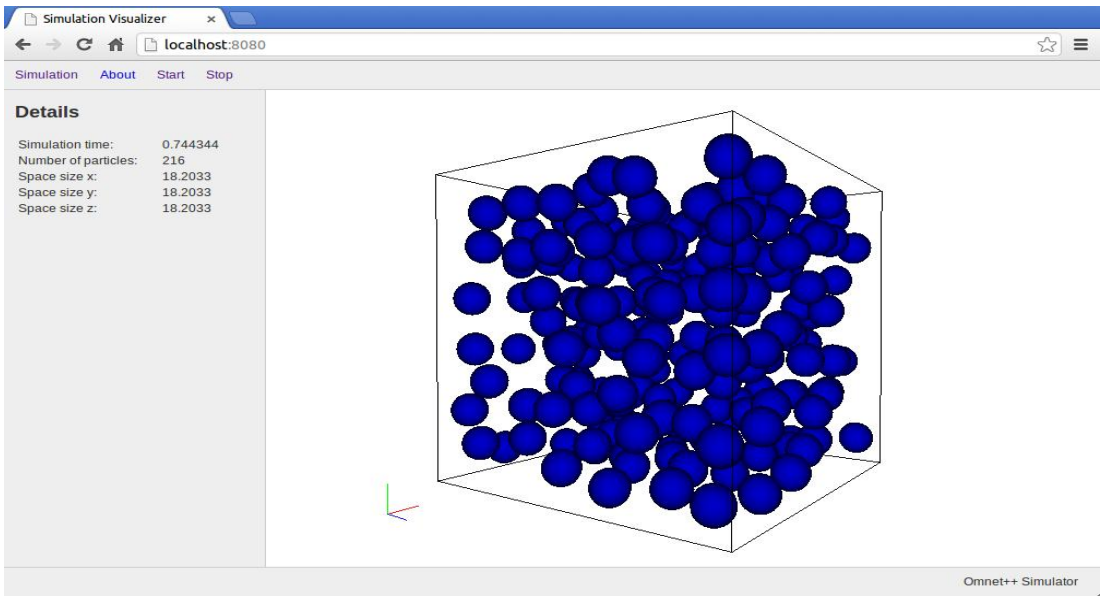

Fig. 3. Chromium web browser visualizing a running simulation.

A. Initial Particle Distribution

We start with a fixed number of particles n=1000 and a low volume density ρ~1%. First we place the particles randomly over a sphere surface to minimize the neighbor particles. Secondly we place all the particles near the center of the simulation to increase it. Finally we repeat the simulation placing the particles following a cube pattern. In Fig. 4 we see that the centered particle distribution initially produces a much higher amount of particle collisions to later tend to a stationary state, similar to the sphere surface distribution, which corresponds to what is expected.

Now we increase the number of particles to n=10000 and the volume density up to ρ=35%. This means that the sphere surface distribution cannot be used in the same way since there is not enough room. We opt to place the particles over a series of concentric sphere surfaces, which for higher volume densities it tends to a close-packing sphere distribution. Fig. 5 shows the projection on a x-y plane of the particles position for the cube distribution (left) and the sphere or close-packing distribution (right).

This time we focus on both the number of collisions per second at the start and end of the simulation, and the initial and final particle distribution. In Fig. 6 we see how the two different initial distributions rapidly tend to stabilize at a certain rate of collisions. Fig. 5 shows the particle distribution

at the start of the simulation and Fig. 7 at the end. We see that after a certain amount of time the effect of the initial distribution becomes unnoticeable on the final particle disposition. We also notice that the cube distribution tends much faster than the dense-packed distribution to a steady state.

In Fig. 6 we see how the transient tends to stabilize much faster than the previous results with a lower volume density. Also both distributions tend to the same collision rate. The higher collision rate during the steady state relates to the increased volume density, which reduces the time between particle collisions.

In Fig. 7 we see the position of the particles centroid after t=100ns for both initial distributions. The first half shows the final position using the cube distribution, whereas the second half shows the final position using the dense-packed distribution. We see how different initial states tend to a similar steady state distribution, as expected.

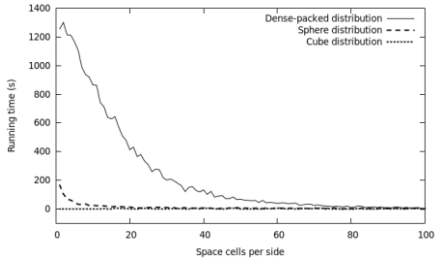

Fig. 4. Particle collisions vs. simulation time for three different initial particle distributions (n=1000 particles, ρ~1%).

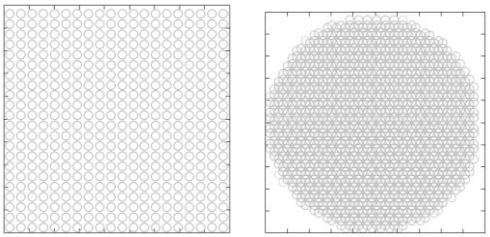

Fig. 5. Left: x-y projection of particles position at t = 0 ns, placed following a cube pattern. Right: x-y projection of particles position at t = 0 ns, dense-packed at the center of the simulation space.

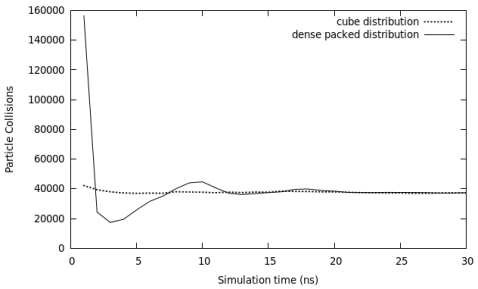

Fig. 6. Particle collisions vs simulation time.

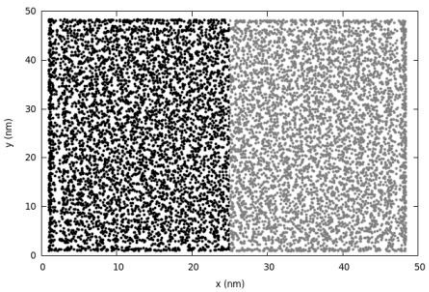

Fig. 7. Comparison of the final particles position between the cube distribution (first half) and the dense-packed (second half), after 100 ns.

In Fig. 8 we plot the paths of a particle during the transient state when using the cube distribution (on top) and the path that follows a particle using the dense-packed distribution (bottom). We see how the last part of the particles path describes the same behavior (similar to a random-walk), according to the steady state. On the other hand, at the beginning of the paths the two particles show a different behavior according to the initial distribution used (cube and dense-packed, respectively). The first one (on top) shows a uniform collision-free paths between sections with higher collisions, whereas the second one (bottom) has a main section with much more collisions that force the particle to remain near the same place for longer time.

Since the cube distribution reaches the steady state faster and minimizes the effect of the transient-state we opt to use it in order to evaluate the algorithm performance.

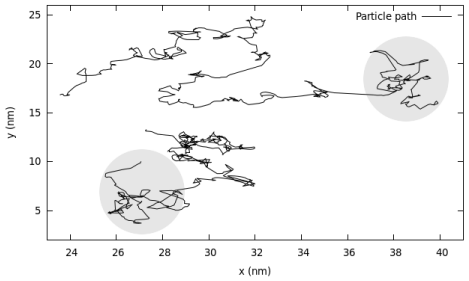

Fig. 8. Particles path for the cube distribution (top) and dense packed distribution (bottom). Shadowed regions show the end of the path.

B. Algorithm Running Time

We run both algorithms varying the number of particles from 1000 to 27000 with a fixed volume density, keeping it at ρ=15%. We also keep the cell size fixed at its lower limit, equal to the diameter of the particles. The particles are placed in the simulation space following a cube pattern. In Fig. 9 we plot the running time versus the number of particles in the system.

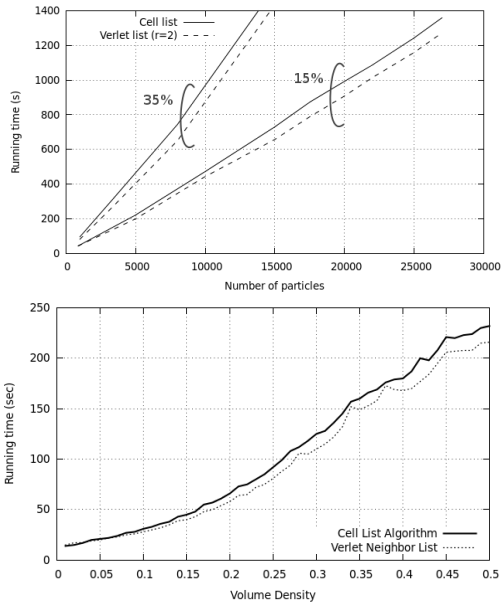

Fig. 9. Top: running time vs. number of particles of each algorithm for a volume density of ρ=15% and ρ=35%. Bottom: running time vs. volume density.

In Fig. 9 (top) we see that execution time grows linearly with the amount of particles in the system. This behavior is

achieved thanks to the use of the cell list algorithm. Furthermore, in both figures we see that the Verlet list algorithm outperforms the previous one. However, we also see that for lower volume densities there is no overall benefit from choosing the Verlet list implementation over the cell list, since the costs of building and updating a neighbor list is higher for less neighbor particles. We must also note the higher slope when using a volume density of ρ=35%. This is due to the higher collisions that the algorithm needs to check.

Now we run the algorithms varying the volume density while keeping a fixed number of particles n=1000 and a Verlet list radius of r=2 (the list radius also equals the space cell side length which is the smallest possible, that is, the diameter of the particles). Again, the initial position of particles follows a cube pattern, and the space cell size is set to equal the diameter of the particles. In Fig. 9 (bottom) we plot the running time versus the volume density of the system. We see that the Verlet neighbor list algorithm performs slightly better than the cell list algorithm in terms of running time, as expected.

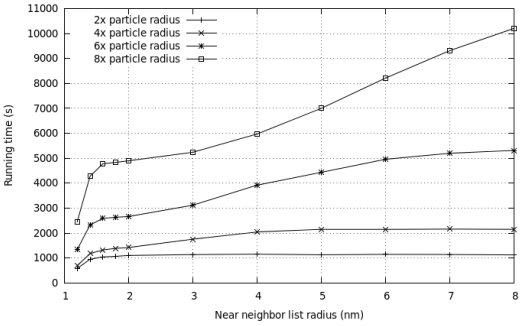

Fig. 10. Comparison between the running time vs. near neighbor list radius (Verlet list radius) using different space cell lengths (n=10000 particles, volume density 35%).

Given the previous results, now we fix the number of particles n=10000 and the volume density ρ=35% and plot in Fig. 10 the running time vs. the Verlet list radius when using different values for the space cell side length. We clearly see that as we increase the length of the space cell we can highly reduce the running time of the simulation by using smaller Verlet radius, up to a limit.

Fig. 11 shows a similar result, but this time comparing the running time against the number of cells per side. We see how there is a tradeoff between the use of different Verlet list radius and the size of the space cells. However, it is convenient to have a combination of both to improve the running time of the algorithm.

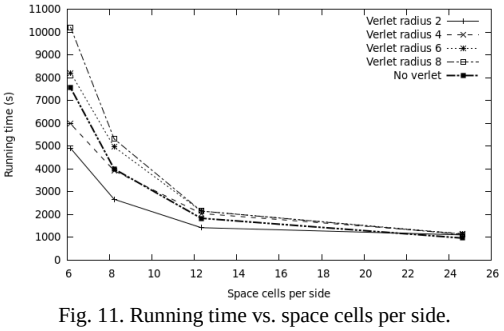

Fig. 11. Running time vs. space cells per side.

Finally, in Fig. 12 we run the algorithm using similar parameters to those used in [8], that is: n=50000 particles and ρ=15%. We see that, despite the higher running time values obtained due to the simulation time used, the overall behavior of the algorithm is the same, indicating that the simulator is well behaved. Herir Sigurgeirsson, Andrew Stuart and Wing-Lok Wan in 2001 [8] report that for a 5000 particle system, their algorithm handles about 16,000 collisions per second on a Pentium III PC. For the same amount of particles our algorithm handles around 30,600 collisions per second using the cell list algorithm. This increase is mainly due to the use of a newer CPU with a higher clock rate. However, the use of a combination of both algorithms improves the previous value up to 34000 collisions per second.

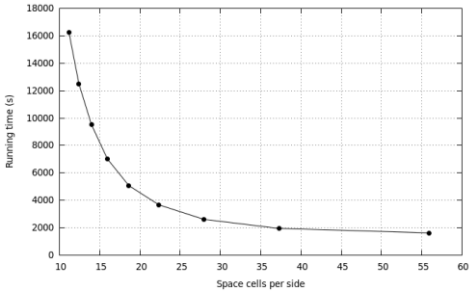

Fig. 12. Running time vs. number of space cells per side.

C. Cell Signaling

In Fig. 13 we reproduce the signaling process between two cells modeled as spheres. The first one is configured as a particle emitter, emitting particles with an emission rate of 1,000 particles/ns to all directions. Then a nearby sphere, configured as a particle receiver, receives the emitted particles removing them from the simulation space in the process. As we can see, the emitted pulse is highly reduced in amplitude mainly due to the distance between the emitter and receiver, proportional to r^2. Also the duration of the pulse received has been increased compared to the emitted pulse due to the particles propagation and collisions.

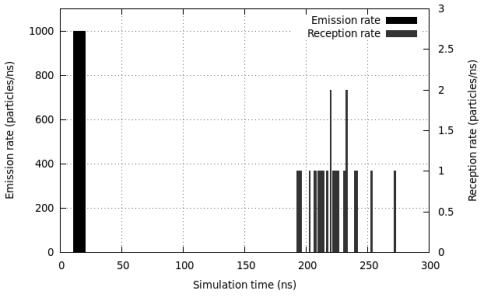

Fig. 13. Signaling process between a particle emitter and a particle receiver.

V. Conclusion

In this paper we have shown that both algorithms are capable to run an N-body simulation with a large amount of particles alongside an event-driven network simulator like Omnet++. This leads us into thinking that we are at a good start point to develop a new open, integrated framework for Omnet++. We have also shown a new visualization software developed with 3D capabilities (in parallel with the existing tcl/tk) thanks to a novel web server module and the use of the latest WebGL technology. This makes us think that the Omnet++ framework may need an update with new visualization capabilities, like incorporating The

Visualization Toolkit (VTK).

As part of future work, one of our immediate goals will be to further develop this work and delve into new and more elaborated simulation scenarios regarding the cell signaling communications.

REFERENCES

[1] Omnet++ Open Source Network Simulation Framework. [Online]. Available: http://www.omnetpp.org/

[2] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed., New York, McGraw-Hill , 2000.

[3] S. M. Ross, *Simulation*, Academic Press, 3rd ed. 2001.

[4] J. Banks, J. S. Carson, and B. N. Nelson, *Discrete-Event System Simulation*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1996.

[5] N3Sim: A Simulation Framework For Diffusion-based Molecular Communication. [Online]. Available: http://www.n3cat.upc.edu/n3sim/

[6] I. Llatser, I. Pascual, N. Garralda, A. C. Aparicio *et al*., "Exploring the Physical Channel of Diffusion-based Molecular Communication by Simulation," in *Proc. Global Telecommunications IEEE Conference and Exhibition*, 2011.

[7] D. C. Rapaport, "The event-driven approach to n-body simulation," *Progress of Theoretical Physics Supplement*, no. 178, 2009.

[8] H. Sigurgeirsson, A. Stuart, and W. L. Wan, "Algorithms for particle-field simulation with collisions," *Journal of Computational Physics*, no. 172, pp. 766-807, 2001.

[9] A. A. Chialvo and P. G. Debenedetti, "On the use of Verlet neighbor list in molecular dynamics," *Computer Physics Communications*, vol. 60, 1990.

[10] Three.js. Open Source WebGL Javascript Library. [Online]. Available: http://threejs.org/

[11] G. S. Fishman, "Monte Carlo: concepts, algotrithms, and Applications," *Springer Series in Operations Research*, Berlin, Germany: Springer-Verlag, 1996.

Daniel Huertas was born in Barcelona. Currently he works on his final project for his Telecommunications Engineering degree at Polytechnics University of Catalonia, Barcelona, Spain.

Alfonso Rojas received the M.S. degree in telecommunication engineering from the Polytechnic University of Catalonia in 1992 and the Ph.D. in telematics engineering in 2000. He is a Tenured Lecturer with the Telematics Engineering Department, Telecommunication Engineering Technical School of Barcelona, Polytechnic University of Catalonia, Spain, and employed since 1994. He has taught several theoretical subjects related with teletraffic, mobile communications, channel coding, cryptography or information theory and also laboratories of communications and simulation. His research interests include engineering education, statistics and data transmission in nano networks.