Pushing the limits
A concept of a parallel microsimulation framework

Author(s):
Charypar, David; Horni, Andreas; Axhausen, Kay W.

Publication Date:
2010-08

Permanent Link:
https://doi.org/10.3929/ethz-b-000156442

Pushing the Limits: A Concept of a Parallel Microsimulation Framework

Date of submission: 2010-08-01

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, CH-8093 Zurich
phone: +41-44-633 35 62
fax: +41-44-633 10 57
charypar@ivt.baug.ethz.ch

Andreas Horni
Institute for Transport Planning and Systems, ETH Zurich, CH-8093 Zurich
phone: +41-44-633 31 51
fax: +41-44-633 10 57
horni@ivt.baug.ethz.ch

Kay W. Axhausen
Institute for Transport Planning and Systems, ETH Zurich, CH-8093 Zurich
phone: +41-44-633 39 43
fax: +41-44-633 10 57
axhausen@ivt.baug.ethz.ch

Words: 6447 + 3 Figures

ABSTRACT

This paper describes the concept of a versatile transport simulation framework to be used for the development of integrated transport simulation models. It is designed to improve modularization and thereby simplify the collaboration between scientists and engineers from different fields. Modularization is believed to be important due to the increasing complexity of the implementation task if models from different areas are to be integrated. In performance critical software projects this complexity is often further increased by the desire or the need for an implementation that can be run in parallel on multiple processors. Furthermore, developing an efficient parallel simulation is not trivial. In addition to the complexity of the modeling task as such one has to deal with communication delays and data availability issues. The idea of the presented framework is to handle this complexity by defining simple rules according to which user developed modules must act. These rules include certain minimum delays between the observation of a change in the system and the triggered reactions, limited vision, and limited traveling speed. To illustrate how the framework is to be used a simples modeling scenario is created and a possible implementation employing the framework is sketched. The example scenario consists of the integration of a pedestrian simulation with a load estimation module and a travel time estimator. Finally, an outlook on the next steps in the development of the framework is given.

INTRODUCTION

In recent years microscopic, dynamic demand and traffic simulations modeling has caught increased attention and found broader use in the field of traffic forecasting and transport planning. One advantage such approaches is that the necessary behavioral models are relatively simple. The reason for this is that they only model directly the behavior of one individual or a small group of individuals (e. g. households). The rest of the behavioral richness is assumed to emerge from the interaction of thousands of such individuals that in the end form a complex system. Onother advantage is and that during analysis it is possible to follow the line of influence down to the individuals, which makes the interpretation of the results more intuitive.

However, microscopic simulations have (at least) one important and well known drawback: for any reasonable sized scenario they are computationally demanding, as each of the individuals naturally must be represented separately.

This in turn makes them relatively inefficient for solving low resolution scenarios (for instance based on coarse zonal data). The reason for this is that during the necessary disaggregation process, a lot of random detail is generated and afterwards simulated. Clearly, this detail does not contribute to the explanatory power of the output and must be interpreted as overhead of such simulation approaches.

On the other hand, if high resolution data is available microsimulations become comparably efficient as, to provide the same precision, any aggregated model needs to refine the zoning to the degree where it fits the level of detail of the input data. Consequently, the resulting OD-matrices become huge as their size grows quadratically with the number of zones.

Overall, it seems to be worthwhile to develop and use microscopic transport models, and they have already been successfully applied to large real world problems. Often, the average work day is under investigation—a problem that can be addressed with equilibrium models. Such modeling challenges can already be handled with available models (e. g. *(1)* even though they require a lot or resources both in time and computing power. One way to reduce the computation time is through parallel execution of simulation programs. While this is becoming more common through multi-core processors that are available nowadays it is still a challenging task to *develop* parallel software. At the same time it is necessary to follow the path of parallelization in future microsimulation software developments to increase the range of problems that can be addressed.

There are, however, different modeling task that cannot be solved using equilibrium models. One important examples is the simulation of unpredictable events like accidents, emergencies, or disasters. Another example is the simulation of longer periods than a single day. Multi-day simulations increase the computation time of microscopic equilibrium models in two distinct ways: First, since the period of interest is longer the simulation of this period naturally also takes longer. Basically doubling the simulated time doubles the computation time. In agent-based microsimulations finding the equilibrium is often approximated by a learning loop of the agents (e. g. in MATSim *(2)* or in TRANSSIMS *(3)*). This loop represents an iterative algorithm converging towards the desired result. The number of iterations necessary to achieve a result of a certain precision naturally depends on the complexity of the solution and hence also on the size of the solution vector: More complex solutions need more iterations to be found. It is clear that finding the equilibrium for a 7-day period is substantially more complex than finding it for a single day.

When combining both considerations above it can be seen that the computational burden of the described iterated approach really becomes en issue. The computation time increases disproportionally with the length of the study period. Consequently, there is a relatively short (computational) limit of what time periods can be investigated using agent-based models with an iteration-type learning loop.

Further more, there exists another argument against iterative learning and against modeling long periods as an equilibrium: In the agent-based context, an (admittedly simplified) interpretation of an equilibrium is that all agents have considered (all) possible choices and found the sequence of actions that maximizes their utility in the given environment. It can be doubted if real people really plan their weeks, months, or even years completely in advance.

Another problem is the assumption that two consecutive runs of the same simulation with only minimal perturbations yield the same result when the list of planned actions is kept the same. One has to remember that complex systems (as the transport infrastructure) tend to amplify disturbances, and this can lead to completely different results at the end. Such effects have been observed in at least one implementation of a microscopic integrated demand model *(4)*.

We believe that it would make much more sense to model longer, multi-day periods as a continuously evolving scenario, where the modeled persons (agents) constantly make decisions on the following time frame. see e. g. Märki *et al. (5)* However, this makes it necessary to make current information about the state of the system available to all objects in the simulation. The online estimation of state variables during a running simulation and consequently the propagation thereof represents a substantial increase in complexity of the simulation. In iterative frameworks this information exchanges happens at the end of the iteration where the generated output is analyzed, processed and made available as static information to agents for re-planning.

In the proposed framework online information processing and spreading across the simulation is provided as a service. This makes it possible that entities in the simulation (e. g. agents) can use them for their continuous planning process.

Based on the above line of reasoning, the development of a framework for large continuous spacial microsimulation was started. The purpose of this tool is to encapsulate all necessary complexity for parallelization and spacial information interchange and hide it from the user. Consequently, a module employing the framework will be comparably simple as it will be able to rely on the framework's functionality provided through a clear interface.

The remainder of this paper is structured as follows: The next section discusses related work, after that the proposed framework is specified and an implementation of three basic modules of an integrated agent-based microsimulation is sketched as it could be done using the described framework. Finally an outlook on planned future steps is given, and a discussion concludes the paper.

RELATED WORK

The following is a short overview about other work that was performed either in the field of parallel microsimulations or in the design of frameworks for transport or urban modeling.

Parallel Traffic Simulations

There are numerous examples of parallel implementations of traffic simulations. Barceló *et al.* *(6)* showed a parallel implementation of their microsimulator AIMSUN achieving a parallel speedup of 3.5 when run on 8 processors. The parallelization concept was to make all data globally accessible.PTV's VISSIM traffic microsimulator also has the capability to run in parallel using a multi-threaded concept. *(7)*

Nagel and Rickert *(8, 3)* showed a parallel version of a cellular automaton used for traffic flow simulation in TRANSIMS *(9)*. They used message passing between processors and achieved a speedup of 10 with 32 processors. They reported latency problems due to Ethernet data communications.

There has been some work on parallel queue-based models (e. g. *10, 11, 12, 13)* Using message passing between cluster nodes, the queue-based model presented in *(10, 11)* achieved a speed-up of 32 using 64 CPUs when simulating a peak period. In *(13)* the authors report a parallel speedup of 53 when using 64 processors for simulating a large scenario.

A number parallel implementations of mesoscopic transport models have been presented in the past. METROPOLIS*(14, 15)* is able to simulate large scenarios efficiently by using a parallel implementation based on up to 16 threads. DynaMIT*(16, 17)* does not parallelize the traffic flow simulation itself but uses task parallelization i.e. different modules are run in parallel. Unfortunately this limits the number of usable processors to the number of modules. DYNEMO*(18, 19)* was run in parallel *(19)* by using a message passing technique on 19 CPUs for simulating small scenarios. Larger numbers of CPUs were reported to be inefficient.

Frameworks for Integrated Modeling

Ferreira *et al.* *(20)* present a framework (MAS-T2er) for integrated multi-agent systems. Their focus is on control strategies and intelligent transport systems. The intended use of their software is "...for cooperative design, visualization and engineering, allowing for the cooperative decision-making by different traffic and transport experts". Their framework is designed to run on distributed systems. It is still under development.

The goal of UrbanSim*(21)* is to model and simulate urban development by modeling the interactions of many different actors that make decisions in the markets for land, housing, non-residential space and transportation.

The Multi-Agent Transport Simulation Toolkit (MATSim-T) *(2)* is a simulation framework for modular development of an integrated transport simulation for large-scale applications. Several modules that where written for/in MATSim run in parallel, e. g. certain versions of the traffic flow simulator(e. g. *13)*, the processing of simulation events, and the activity planning module. However, the program code of the framework itself is executed sequentially.

FRAMEWORK

In this section first, the problematics that necessitate the creation of the described programming framework are discussed, second, the model concept is derived from these problematics, and third, the design of the software is elaborated on.

Motivation

The context of this work is the microsimulation of different aspects of travel. Microsimulation can be a very powerful tool to gain insight into travel behavior, emerging dynamics of systems with human actors, and effects resulting from external measures. Unfortunately, microsimulation models are computationally demanding, which makes it hard to apply them to sufficiently large problems. Consequently, a tool that accelerates the development and execution of microsimulation models would increase their range of application.

Integrating different models in one more complex microsimulation also widens the range applications: It enables researchers to investigate interactions between different aspects of the modeled scenario. For instance integrating a traffic simulator and a routing module with a location choice module makes it possible to study the effect of congestion on location choice. To keep the modeling task as simple as possible it is desirable to have a high level of modularization and to have the development process of modules as isolated as possible. This also helps to control code complexity.

In many research projects emerging phenomena (e. g. urban gridlock) are of special interest. Such emergence naturally only occurs in sufficiently large systems, and hence researchers must be able to cope with such systems computationally. Here, being able to efficiently use parallel computers might make the difference if interesting effects might be investigated or not.

Unfortunately, until now developing parallel programs tremendously increases the complexity of a coding project and makes it hard to handle.

The classical modeling approach in transport planning is to compute a user equilibrium to investigate long term effects of changes. Recently the immediate response to unexpected events is caching more and more interest. To be able to model such effects, users of the system (i. e. agents) must have access to estimates of the current state of the system. When developing a simulation software, including online state estimation corresponds to integrating another module. Obviously this further increases software complexity.

Based on the above considerations it seems that a framework solving the described problems with code complexity while at the same time simplifying parallel and modular programming would be very useful. It would facilitate further research in the field of integrated transport microsimulations.

Concept

The main objective for the described framework is to create a tool that simplifies the modularization of a complex transport modeling/simulation project and to reduce the code complexity of the modules at the same time. This is achieved through taking over the tasks of code parallelization and distribution of the workload on different computers/processors, and information distribution and interchange between different entities of the simulation.

To enable the collaboration between user-developed modules and the framework, respectively it is necessary to specify the cut line between their individual fields of responsibility. One of the key aspects of the presented framework is to limit the vision and motion capabilities of object in the simulation. This is done to avoid a problem that otherwise often occurs when a simulation program is later enhanced to run an a parallel computer. The following paragraph should illustrate that problem

When developing a simple simulation program (i. e. at the beginning of the coding project), initially the visibility of information about spatially distributed objects is usually assumed to be global, for the sake of simplicity. For example when developing a car following model the speed and position of all cars on a road might be stored in an array representing the state of the system, and this state is stored at one specific spot in computer memory. As the model grows, larger simulation are being performed and the desire for a parallel implementation arises. Often distribution across multiple threads is tried here first, unfortunately with limited speedup of the simulation. The problem is that all objects in the simulation access the same data set (the array described above) and also make changes there. As a result this part of the simulation becomes a bottleneck, essentially slowing down the simulation to single-CPU speed.

The underlying problem is, that all data is visible globally and instantly in the simulation. As a result, when a data point is changed by one processor this new information must be propagated to all other processors before they can continue with their individual tasks. Since communication speed between processors is physically limited the simulation essentially stalls until the message has been propagated.

Our approach to this problem is to limit the assumed visibility of information and the speed of its propagation and hence give the processors more time to synchronize the state of their memory. This avoids stalling the CPUs and hence improves the simulation speed.

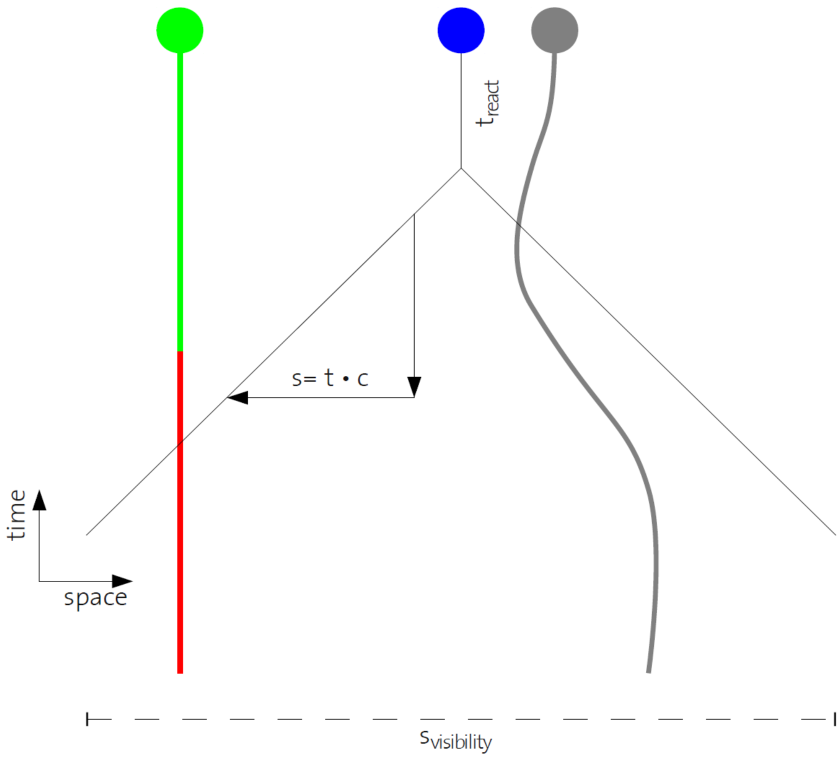
The first such constraint is *limited visibility of information*. To reduce the amount of data that must be held readily available to an observer, we assume a maximum radius of vision. This radius can be chosen by the user at the beginning of the simulation run. It will depend to a great extent on the problem at hand. In case of a pedestrian simulation for evacuations of buildings it might be set to e. g. 20 meters. In another example, where we want to simulate freeway car traffic, some 500 meters might be more appropriate. After having specified such a visibility radius it is assumed that no module will request or need information from farther away and on the other hand that data provided by the framework is complete inside this radius.

The second introduced constraint is *delayed perception*. One can imagine this as a reaction time. Virtually any system shows delayed reaction to external information. In the case of a car following simulation this might be set to half a second. This is the time from the moment an information can be perceived by some entity to the moment this object can take some action based on it.

The third and maybe most important constraint is *limited speed of motion and information propagation*. If at one point in the simulation there is some change to the state of the system, this change cannot be observed instantly in the whole area around this point. Rather, the information has to travel (at a certain speed) through the system much like a sound wave travels from the source. Observers will not take notice of the change until this *information front* hits them. Similarly, moving objects are not allowed to move faster than a certain maximum speed. While this last constraint can be observed in reality (objects and information cannot travel faster than light), it might seem odd to limit the speed in a simulation somewhat arbitrarily to a relatively low value.

The final set of constraints is illustrated in Figure 1, where the perception of an object (blue) is shown in a space time diagram. $s_{visibility}$ is the visibility range, t_{react} is the reaction time, c is the speed of information propagation, t represents time, and s space.The thin black line represents the front of information available to the blue object. The green object is stationary and changed its state from red to green at a certain point in the past. Since the blue object is relatively far away this change can not yet be perceived and hence the red state is still relevant to the blue object. The gray object is moving and the position where its path intersects the

FIGURE 1 Limited Perception of Information Based on Introduced Constraints



information front is the latest position visible to blue.

One basic concept of our simulation is to map the spatial ordering of the simulated area to the processors and hence to computer memory. In a sense, if information travels through the virtual domain of the simulation, it travels from processor to processor and from memory bank to memory bank involved in the computation. Since we effectively limit the speed of information propagation in the virtual world, the data also travels at limited speed between the involved computers. This in turn increases the achievable computing speed as the reduced requirements are more easily satisfied.

Design

In our framework the simulation domain is subdivided using a uniform grid with cells of side length $s = t_{react} \cdot c$, which is defined during the configuration phase of the framework. This size of the cells was selected as it simplifies the information exchange across processor boundaries.

As a result, the maximum number of processors that can be used is equal to the number of cells used to subdivide the domain. However, it is possible to join multiple cells to use on single CPU, eliminating the need for physical communication between these parts of the simulation. This is desirable if scenarios show differently loaded cells. Cells with comparably little work to do should be joined and assigned to a single CPU while heavily loaded cells should be simulated exclusively on a separate processor.

Internally, the framework holds a comprehensive list of replicated cells for each real cell, that is simulated. These lists form a discretized form of the information front described in Figure 1. Further more it is expanded to represent not only the information for one single point

FIGURE 2 Information Potentially Needed in a Cell During a Certain Time Period

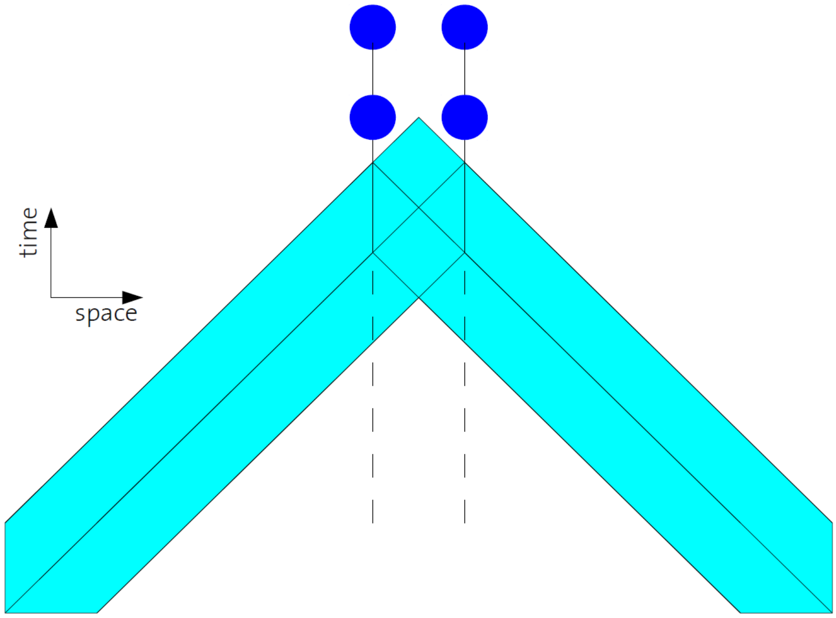
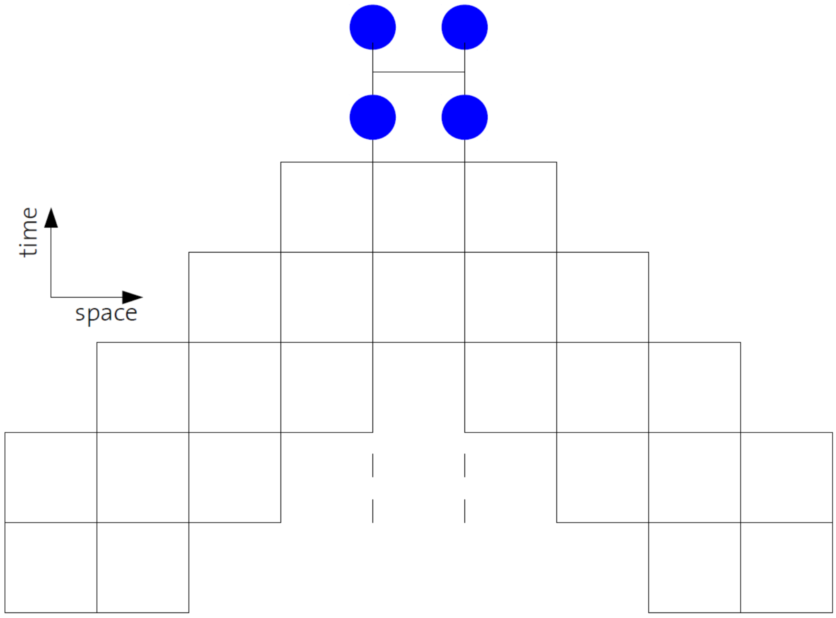


FIGURE 3 Discretized Information Domain in a Cell



but from the union of all data that might be needed by any entity situated in the cell during one time period as long as the reaction time. A graphical representation of this can be found in Figure 2. Further more, to simplify the process of exchanging information between adjacent cells it is useful to discretize the information domain in a similar way as the simulation domain is discretized into square cells. This is illustrated in Figure 3 which shows a discretized form of Figure 2.

The goal of the framework described in this paper is to encapsulate as much complexity associated with information interchange and parallelization, and hide it from application programmers developing a module that is part of a larger integrated simulation. The basic approach is to employ a client-server architecture where both, framework and user modules take a dual role. On the one hand the framework is the server and modules are clients during phases where information about the state of the environment is requested by the modules. On the other hand the framework becomes the client after computations have been completed by user modules and the resulting new information needs to be published to the system which is when modules are in the server role.

It seems to be straightforward to define two interfaces here. One defining how information about the state of the system can be gathered by user-developed modules (the *gathering interface*) and a second interface for the opposite direction of data flow, the *publishing interface* The gathering interface consists basically of a function called with the following arguments:

- **cell_index**: Index of the cell of interest for the current query.
- **start_time**: Start time of the query period.
- **end_time**: End time of the query period.
- **predicate**: A mathematical predicate. True for relevant information.

The function returns a list of information objects that all are relevant in the given cell during the given time period and satisfy the query predicate. Information objects represent published information about simulation entities that might be relevant to other entities. For example, in a car following model this might be information about the current position, speed, and acceleration of a car, but it would usually not contain the route, the destination, or its desired speed of the car. The querying module can assume that no information objects are forgotten and that now new relevant information may become available during the processing of the current time period. It is clear that this is only possible if the current simulation time t_{now}, the reaction time t_{react}, the query start time t_{start}, and end time t_{end} satisfy certain condition:

$$
\begin{aligned}
t_{\mathrm{end}} - t_{\mathrm{start}} &\leq t_{\mathrm{react}} \qquad (1)\\
t_{\mathrm{end}} &\leq t_{\mathrm{now}} \qquad (2)
\end{aligned}
$$

In the other direction of information exchange the interface looks very similar. Each of the simulated entities, and hence the implemented modules must provide a function that can be called by the framework to obtain all information objects. At the end of a simulation time step the framework goes through all cells and for each entity the calls the data providing function with the following arguments:

- **start_time**: Start time of the query period.
- **end_time**: End time of the query period.

The object should react with a list of information objects that describe all publicly available information about the object at hand.

EXAMPLE MODULES

Using a small example, we would like to sketch how the described framework would be employed to solve a real modeling task, involving the implementation of a couple of modules. The scene of interest is a music festival with many visitors that in general spend many hours on the festival venue. During their stay they have to get something to food and drink from time to time. For this purpose there is a number of food stands distributed on the periphery of the area while the main stage can be found near the center. The modeling task is now to simulate (and maybe predict) the movements conducted by pedestrians looking for available food stands. It would not be realistic to assume that the one food stand closest to the stage would have the capacity to serve all visitors in a reasonable time. For the modeling task at hand it is of particular interest how visitors would use more distant and hence less crowded stands to get served in shorter time.

The following is a relatively straight forward example of how the above modeling task might be solved. Certainly it is not very sophisticated and leaves a lot of room for improvements. The proposed realization is meant to be illustrative rather then comprehensive and should give an idea how the framework would be employed.

Clearly, one necessary module is a pedestrian simulation that models how people walk around based on their direct surrounding. One possible choice for the underlying simulation model might be *(22)* as it already implements attraction through other objects that might be used to model how visitors generally want to get as close to the central stage as possible. A second module would have to be added with the task of estimating the current loading of all food stands. Finally, a third module for estimating pedestrian densities would provide a way of estimating travel times to different locations.

For each of the modules it must be now decided what are the simulated entities, what is the information these entities needed about the environment for proper operation, and what is the information generated. For the generated information it is especially instructive to think about for what the information is to be used and what new information will be generated from it in turn. In the case of the pedestrian simulation it is relatively clear that the simulated entities are pedestrians and that they need information about surrounding persons to be able to act an react. Symmetrically, the information pedestrians need to publish is their position, velocity, plus current activity, namely if the are waiting for food or not. This part will be important later in this section. Now, when a pedestrian receives the positions and velocities of all other pedestrians near by (within a range of $s_{visibility}$), it has sufficient information to take its next actions. (E. g. decelerating, avoiding a collision, or changing direction).

When it comes to the pedestrian density estimator a simple grid based approach is used in this example. The simulated entities are nodes an a square lattice with a spacing of one visibility range of a pedestrian. Each node uses the gathering interface to get the positions of all pedestrians near by. From this the node can easily compute a local density estimate using a kernel method for instance. It is clear that at this point in time the node cannot know anything about pedestrian densities outside the visibility range. For this reason it is essential that this local estimate is published by the node. In the next turn, each node not only gets the information about pedestrians near by but also the (local) density estimates of neighboring nodes. By storing them, the node can construct a density map of an area larger than the visibility range. In the next turn, this whole density map is published through the framework, providing information to nodes even farther away By iterating this procedure, very soon each node will possess an estimate of the pedestrian densities in the whole simulated domain. This is actually a density map and can be used e. g. for routing and travel time estimation.

The food stand loading estimator (FSLE) has as similar task as the density estimator. For this reason its design is similar and it also operates in a similar way. The FSLE is also designed using nodes. Each node holds a list of all food stands in the domain and how many visitors are currently near them waiting for food. This list is initially empty and filled with information as the simulation progresses. In a first step the loading of local (to the node) food stands is estimated from the pedestrians' positions and their current activity. The value is stored in the list. In the next step, each node takes its updated list and publishes it using the communication framework. Then the new information is collected once again through the gathering interface and the estimates from neighboring FSLE nodes are merged into the current list of food stand loads. At this point the list already contains more information. In each turn, the content of the food stand loading list grows, until each FSLE node has a complete list of food stand load estimates.

At this point all parts of our simulation are ready to be used. We would like to show now the steps taken if a pedestrian in the simulation becomes hungry and hence wants to find an available food stand: In the first step it gets the last version of the food stand loading list published by the nearest FSLE node. This functionality is made available by the data gathering interface. Now, for each food stand that is no overcrowded, a travel time is estimated using the last published pedestrian density map. This data is also received through the data gathering interface of our framework. Finally, the pedestrian selects the best of the available choices, considering expected waiting time at the food stand, travel times, and travel distances. The pedestrian starts to walk towards the selected food stand, thereby reacting to all other pedestrians that he encounters.

FUTURE WORK

The described framework is still in a relatively early state of development. The implementations must be tagged as prototypes and the interfaces are not yet finalized. One of the aspects we are still working on is how information objects are transported from one simulation cell to the others. There are different paradigms that one can follow here. In general one has to trade off between communication and processing overhead. One extreme is the very sophisticated selection of only the bare minimum of information that needs to be transfered between processors to guarantee correct results. This obviously comes at the expense of spending a lot of time evaluating the necessity of a data point. At the other and of the scale stands the preference for quick checks selecting a relatively large volume of data for transfer to other cells. Hence, using this design, processor loads will be relatively low while communication demands will be high.

The next steps will be to finalize the communication paradigm based on performance and complexity considerations, finalize the interfaces to user modules, and then create a first published version of our framework. Since the presented software is meant to ease collaboration on integrated transport modeling projects we seek cooperation with other interested researchers that would like to implement modules using the framework. Consequently, the software is meant to be released to public domain.

We are currently working on one first project employing our parallelization framework. In that project we are aiming to simulate periods of more than 30 days in an integrated agent-based environment. A special focus is the activity planning process, especially the resulting weekly rhythms and effects of business-holidays on infrastructure usage. The activity planning module is currently under development and shows first promising results *(5)*. Apart from activity planning there are other modules necessary for the functioning of this integrated simulation: For the adaptive creation of routes there is a on-line travel time estimator under development. Furthermore, we envisage a location choice module based on current load factors to represent the flexible choice of shopping and leisure locations.

Apart from getting interesting insights into modeling and simulation of multi-day periods, we plan to measure the effectiveness of our framework by testing different scenario size with various numbers of processors for parallelization. If our approach proves to be right, we should be able to demonstrate good scaling of performance with the number of processors.

DISCUSSION AND CONCLUSION

The concept of a parallel framework to be used in the development of integrated transport microsimulations was presented. It should increase and ease the cooperation between researchers and engineers from different fields by allowing for strict modularization of different model parts.

The framework takes care of the complex tasks of parallelization and information exchange between modules. In the experience of the authors these are often critical parts of integrated simulations which often lead to problems in performance, reproducibility of results, and stability of the over all software package. By assuming a two dimensional domain for all modules (this can be easily extended to three dimensions) and by setting explicit limits on what actions can be performed (maximum speed of motion), how quickly they can be perceived (reaction time), and how far away they are visible (visibility radius) it becomes possible to confine the tasks of parallelization and information exchange in the described framework.

It must be noted that the introduced limits might also produce problems in certain cases. Many models implicitly assume global availability of information and it is not clear in advance if and how they can be fit into the described framework. Also some modules might not have an obvious spatial interpretation. However, the authors believe that mapping everything to 2D space and assuming a certain delay in the propagation of information does not represent a real problem in all but very few cases.

In the example shown it was demonstrated how the framework would be used for a simulation of intelligent pedestrians performing food stand location choice based on load estimates and route calculation through crowded areas. Employing the framework in this example was shown to be straightforward.

The current prototype implementations of the framework show reasonable parallel performance. However, future test will have to show the performance when simulating real integrated models. There certainly will be an overhead involved with our framework, as there always is when introducing a layer of abstraction. However, we believe that the benefit through improved efficiency in the development of integrated simulations will by far exceed the moderate simulation overheads.

REFERENCES

1. Balmer, M., A. Horni, K. Meister, F. Ciari, D. Charypar and K. W. Axhausen (2009) Wirkungen der Westumfahrung Zürich: Eine Analyse mit einer Agenten-basierten Mikrosimulation, *Final Report*, Baudirektion Kanton Zurich, IVT, ETH Zurich, Zurich, February 2009.

2. Balmer, M., M. Rieser, K. Meister, D. Charypar, N. Lefebvre and K. Nagel (2009) MATSim-T: Architecture and simulation times, in A. L. C. Bazzan and F. Klügl (eds.) *Multi-Agent Systems for Traffic and Transportation Engineering*, 57–78, Information Science Reference, Hershey.

3. Rickert, M. and K. Nagel (2001) Dynamic traffic assignment on parallel computers in

TRANSIMS, *Future Generation Computer Systems*, **17** (5) 637–648.

4. Rieser, M. and K. Nagel (2008) Network breakdown “at the edge of chaos” in multi-agent traffic simulations, *The European Physical Journal B - Condensed Matter and Complex Systems*, **63** (3) 321–327.

5. Märki, F., D. Charypar and K. W. Axhausen (forthcoming) Continuous activity planning for a continuous traffic simulation, paper presented at the *90th Annual Meeting of the Transportation Research Board*, Washington, D.C.

6. Barceló, J., J. L. Ferrer, D. Garcia, M. Florian and E. Le Saux (1998) Microscopic traffic simulation, in P. Marcotte and S. Nguyen (eds.) *Equilibrium and Advanced Transportation Modelling*, chap. 1, 1–26, Kluwer, Dordrecht.

7. PTV America (2010) PTV America, webpage, `http://www.ptvamerica.com`.

8. Nagel, K. and M. Rickert (2001) Parallel implementation of the TRANSIMS micro-simulation, *Parallel Computing*, **58** (2) 1611–1639.

9. Nagel, K., D. E. Wolf, P. Wagner and P. M. Simon (1998) Two-lane traffic rules for cellular automata: A systematic approach, *Physical Review E*, **58** (2) 1611–1639.

10. Cetin, N. (2005) Large-scale parallel graph-based simulations, Ph.D. Thesis, ETH Zurich, Zurich.

11. Cetin, N., A. Burri and K. Nagel (2003) A large-scale multi-agent traffic microsimulation based on queue model, paper presented at the *3rd Swiss Transport Research Conference*, Ascona, March 2003.

12. Charypar, D., K. W. Axhausen and K. Nagel (2007) An event-driven queue-based traffic flow microsimulation, *Transportation Research Record*, **2003**, 35–40.

13. Charypar, D., M. Balmer and K. W. Axhausen (2009) High-performance traffic flow microsimulation for large problems, paper presented at the *88th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2009.

14. Marchal, F. (2001) Contribution to dynamic transportation models, Ph.D. Thesis, University of Cergy-Pontoise, Cergy-Pontoise.

15. de Palma, A. and F. Marchal (2002) Real cases applications of the fully dynamic METROPOLIS tool-box: An advocacy for large-scale mesoscopic transportation systems, *Networks and Spatial Economics*, **2** (4) 347–369.

16. Ben-Akiva, M. E., M. Bierlaire, H. Koutsopoulos and R. Mishalani (1998) DynaMIT: A simulation-based system for traffic prediction, paper presented at the *DACCORS Short Term Forecasting Workshop*.

17. DynaMIT (2006) Intelligent transportation system program, webpage, `http://mit.edu/its/dynamit.html`.

18. Schwerdtfeger, T. (1984) DYNEMO: A model for the simulation of traffic flow in motorway networks, in J. Volmuller and R. Hamerslag (eds.) *Proceedings of the Ninth International Symposium on Transportation and Traffic Theory*, chap. 4, 65–87, VNU Science Press, Utrecht.

19. Nökel, K. and M. Schmidt (2002) Parallel DYNEMO: Meso-scopic traffic flow simulation on large networks, *Networks and Spatial Economics*, **2** (4) 387–403.

20. Ferreira, P. A. F., E. F. Esteves, R. J. F. Rossetti and E. C. Oliveira (2008) A cooperative simulation framework for traffic and transportation engineering, in L. Yuhua (ed.) *Cooperative Design, Visualization, and Engineering*, vol. 5220 of *Lecture Notes in Computer Science*, 89–97, Springer, Berlin.

21. Waddell, P. (2002) Urbansim: Modeling urban development for land use, transportation, and environmental planning, *Journal of the American Planning Association*, **68** (3) 297–314.

22. Helbing, D. and P. Molnár (1995) Social force model for pedestrian dynamics, *Physical Review E*, **51** (5) 4282–4286.