A Message-Based Framework for Real-World Mobility Simulations

Christian Gloor, Duncan Cavens, and Kai Nagel

Abstract. There is considerable interest in the simulation of systems where humans move around, for example for traffic or pedestrian simulations. Any such simulation consists of two layers: the simulation of the "physical" system, which includes effects such as interaction with other agents or the environment; and the simulation of the "mental" layer, which generates strategies of the agents. The traditional way to couple the modules is to use files. The disadvantage of that approach is twofold: The computational performance is limited by I/O; and the modules can only be run sequentially.

In order to overcome these problems without sacrificing modularity, a message-based approach is presented. Agent strategies are sent via messages to the simulation of the physical system, which executes them and sends back performance information in the form of "events". The strategic modules listen to these events, memorize them in some appropriate way, and possibly generate revised strategies. These strategies are sent to the simulation of the physical system immediately, so that the representation of the agent in the physical system will switch to the new strategy right away.

In addition, the same messages can also be used to plug helper modules, such as viewers or recorders, into the system. An implementation of the framework is tested within our project, which explores the feasibility of using autonomous agent modeling to evaluate future scenarios in a tourist landscape in the Swiss Alps.

1. Introduction

As many planning problems focus on processes that evolve over time in a complex environment, it is often difficult to evaluate the long term implications of a planning decision. Computer simulations can be used as a method for evaluating proposed future scenarios in planning [18]. However, most simulation efforts in spatial planning have focused on large spatial scales (such as at the city and regional levels) and on relatively abstract concepts (such as land use patterns, traffic and economic development), while one can argue that the planning decisions that have the most impact on individual citizens tend to be either at a relatively small scale or have very local impacts.

Multi-agent simulations, where each agent is modeled individually, allow to look at this problem from a point of view of a person walking in an area. We developed a large scale pedestrian simulation which is intended for the simulation of hikers in the Alps [1, 8, 9], but which should also be applicable to related problems such as urban

park design, building design, evacuation simulation [6], department store design, etc. The same computational techniques will be applicable for any kind of distributed multi-agent mobility simulation, in particular for multi-agent traffic simulations.

Any such simulation system does not just consist of the mobility simulation itself, but also of modules that compute higher level strategies of the agents. For traffic and for hiking simulations, the most typical of these modules are: (i) demand generation; (ii) route generation. The demand generation module generates demand for movement between different locations (trips); the routing module computes the actual paths that these trips will follow.

A major and very important difference to the traditional approach is that it is now possible to make all the modules completely microscopic on the level of the hikers. Microscopic means that in all modules each individual agent retains its identity, including, for example, gender, age, income, physical fitness, or remaining energy level. It is, we hope, easy to see how such information can be used for better modeling.

Traditional implementations of transportation planning software, even when microscopic, are monolithic software packages [15, 3, 13, 16]. By this we do not dispute that these packages may use advanced modular software engineering techniques; we are rather referring to the user view, which is that one has to start one executable on one CPU and then all functionality is available from there. The disadvantage of that approach is twofold: All the different modules add up in terms of memory and CPU consumption, limiting the size of the problem. And second, although the approach is helpful when starting as one software project, it is not amenable to the coupling of different software modules, developed by different teams on possible/different operating systems.

A first step to overcome these problems is to make all modules completely standalone, and to couple them via files. Such an approach is for example used by TRANSIMS [19]. The two disadvantages of that approach are: (1) The computational performance is severely limited by the file I/O performance. (2) Modules typically need to be run sequentially. Each module needs to be run until completion before starting the next module. For example, the routing module can only be run before or after the mobility simulation. This implies that agents cannot change their routes while the mobility simulation is running.

Another possibility is to use a database system for information exchange between modules. This is easiest to imagine if modules still run sequentially. Then each module changes the state of agents in the database, and some central schedule decides which module to run at which time. This approach is, for example, used by URBANSIM [20].

The approach presented in this paper is to couple the modules by messages (Fig. 1). In this way, each module can run on a different computer using different CPU and memory resources, which overcomes the memory bottleneck of the monolithic approach. The approach also avoids the bottleneck of file I/O, since data is not written to file at all while the simulation is running. Finally, the message-based approach allows real-time interaction between the modules: for example, if an agent is blocked in congestion, the strategy generation modules can react to this new situation and submit, say, new routes or activities while the agent is still en-route.

On simulations with tens of millions of agents, issues such as bandwidth usage, packet loss, and latency become increasingly important. As a result, we use different

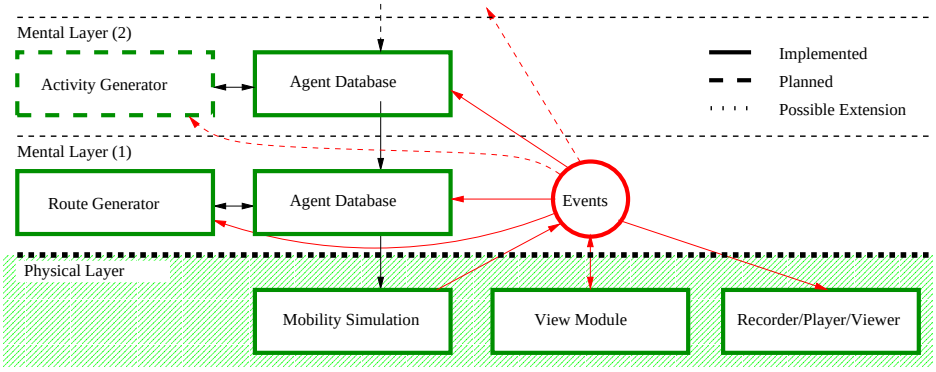

FIGURE 1. Any simulation system does not just consist of the pedestrian simulation itself (physical layer), but also of modules that compute higher level strategies of the agents (mental or strategic layers).

network protocols and implementations tailored to specific requirements of inter-module communication. This paper will also discuss some of these protocols, and the diverse purposes they serve in a distributed multi-agent simulation.

2. The Framework

As said above, our approach is to model each tourist individually as an “agent”. A synthetic population of tourists is created that reflects current (and/or projected) visitor demographics. These tourists are given goals and expectations that reflect existing literature, on-site studies, and, in some cases where sufficient data is not available, are based on experts’ estimates. These expectations are individual, meaning that each agent could potentially be given different goals and expectations.

These agents are introduced into the simulation with initial plans (see later), but with no “knowledge” of the environment. The agents execute these plans, receiving feedback from the environment as they move throughout the landscape. At the end of each run, the agents’ actions are compared to their expectations. If the results of a particular plan do not meet their expectations, on subsequent runs the agents try different alternatives, learning both from their own direct experience, and, depending on the learning model used, from the experiences of other agents in the system.

After numerous runs, the goal is to have a system that, in the case of a status quo scenario, reflects observed patterns in the real world. In this case, this could, for example, be the observed distribution of hikers across the study site over time.

A “plan” can refer to an arbitrary period, such as a day or a complete vacation period. As a first approximation, a plan is a completely specified “control program” for the agent. It is, however, also possible to change parts of the plan during the run, or to have incomplete plans, which are completed as the system goes.

2.1. The physical layer (mobility simulation)

In our architecture, agents' plans are submitted to the mobility simulation. The mobility simulation executes all plans simultaneously, computing interactions of the agents with the environment and with each other. For example, if two agents want, according to their plans, to be at the same place at the same time, the physical interaction of the simulation will prevent that and compute physically plausible solutions instead.

Information about each agent's performance is sent back to other modules in the form of events. Examples of events are "agent left hotel", "agent entered link", "agent had nice view", etc. Events come together with a time stamp and the agent id number.

Many modeling techniques exist for the simulation of pedestrian movements. For our simulations, we need to maintain individual particles, since they need to be able to make individual decisions, such as route choices, throughout the simulation. This immediately rules out field-based methods, where particles are aggregated into fields. We also need a realistic representation of inter-pedestrian interactions, which rules out mesoscopic models, such as queue models or smooth particle hydrodynamics models.

For microscopic simulations, there are essentially two techniques: methods based on coupled differential equations, and cellular automata (CA) models. In our situation, it is important that agents can move in arbitrary directions without artifacts caused by the modeling technique, which essentially rules out CA techniques. We therefore use a coupled differential equation model for pedestrian movement (social force model [10]).

An important functionality of our simulation is to evaluate the visual quality of the landscape. It turns out that this can be done by using the 3d visualizer that is also used for humans to interact with the computer system (see below). However, instead of displaying a view on the screen after it is computed, the video memory is read out in order to determine what individual agents "see" as they move through the landscape. The agents' field-of-view is analyzed, and events now contain information describing what the agent sees. Depending on the needs of the brain modules, these events either list all of the individual objects (houses, restaurants, forest stands, or individual trees) or return synthesized information about particular visual metrics (such as enclosure, percentage of view that is non-vegetative, etc.)

2.2. Learning

Initially, every agent starts with a plan that, in its opinion, fulfills its expectations. For example, if the period of interest is a day, then such an initial plan might refer to a specific hike. To do this, the agent chooses *activity locations* it wants to visit, like hotel, peak of mountain, restaurant etc. The chain of activity locations is then handed over to the *routing* module, which calculates the routes between activities according to the information available. This information can be static and global, like shortest path information based on the street network graph.

The mobility simulation then executes the routes. The agent experiences the environment and sends its perception as events to the other modules.

From here on, the system enters the *replanning* or *learning loop*. The idea, as mentioned before, is that the agents go through the same period (e.g. day) over and over again.

During these iterations, they accumulate additional information, and try to improve their plan.

The two critical questions are (1) how to accumulate, store, and classify that information, and (2) how to come up with new plans. Both questions are related to (artificial) intelligence, and we are certainly far away from answering them in their entirety. Nevertheless, our system contains the following elements which makes it able to learn.

As one can see in Fig. 1, there are **agent databases** associated with each level of the planning hierarchy (e.g. activities, routes). The task of these agent databases is to store plans and to accord scores to them. That is, every time an agent comes up with a new plan, that plan is added to the repertoire of plans. In addition, the agent databases listen to the events emitted by the mobility simulation, and use these events to calculate a score for each plan once it has completed. If an agent database module assigns a bad score to a simulated hike, it tries to avoid its elements in future hikes. If it gets good feedback, however, it will try to reuse the elements of a hike, and combine these into a new hike, which will be simulated and scored again.

The agent databases, at each level of the hierarchy, need to obtain new plans from time to time. Such plans can be constructed by the following methods:

- New plans can be constructed from global information.
- New plans can be constructed from information that an individual agent has accumulated.
- New plans can be constructed by randomly modifying an existing plan. In the language of genetic algorithms, this is called (generalized) **mutation**.
- New plans can be constructed by taking two existing plans and combining elements of them. In the language of genetic algorithms, this is called (generalized) **crossover**.

An example of a method that generates elements of plans from scratch based on global information is a **shortest (fastest, best) path algorithm**. This algorithm takes starting and ending location as input, and computes the best path connecting those two as output. An example of such a method is a Dijkstra algorithm [4]. It is possible to use generalized cost functions in such a router, which makes it possible to find different solutions for different types of agents. For example, a physically fit person might prefer a route that is steeper, while a physically less fit person might get a route that includes more possibilities to rest. It is also possible for the router to listen to the streams of events, for example finding out how many agents are on which links as a function of the time-of-day. This makes it possible to include congestion effects.

An example of a method that generates elements of plans from individual agent information is a module based on a mental map (e.g. [2]). This module listens to the stream of events. However, rather than scoring complete plans as the agent database does, it constructs a mental map of the spatial environment. From the times when agents enter and leave links, the mental map learns how long it takes for an agent to walk along each link in the network. Also, the mental map accumulates all the other events that occur on every link. Using these values for each link in the network, a router based on this mental

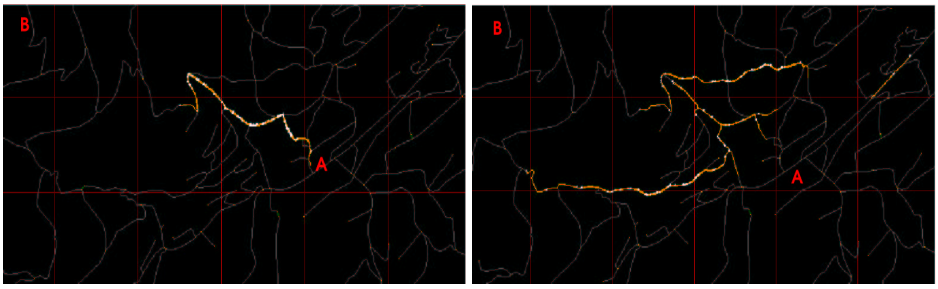

FIGURE 2. Agents leave the hotel (A) in the morning and hike to the mountain peak (B). On the first run, where no hiker knew about the others' intentions (top), and the situation after 50 iterations.

map is able to return a route for each agent, based on its expectation and demographic data. A prototype implementation of such an approach can be found in Ref. [11].

It is possible to make the mental layer module dynamic. In this case, it observes the agent on its path through the virtual environment. As soon as it detects an option that might yield a better score than the current plan, e.g. using a better path, or entering a restaurant, it notifies the agent in the simulation. Using this mechanism, agents are able to react to unpredicted changes in the environment, like weather changes or congestion.

It should be noted that the distinctions between these modules are not sharp. For example, an agent database may run out of memory if it memorizes as separate entities plans that differ only in small details; in that case, the agent database might have to start building a mental map of the world. In this case, it becomes similar to the mental map module as described above.

Fig. 2 shows an example of agent learning. Both pictures show a snapshot of the same region at the same time of day. All agents leave the hotel (A) in the morning, and hike to the mountain peak (B). On the first run (top), when the hikers do not know about the other hikers' intentions, all hikers take the same path, which they consider the best. After 50 iterations, agents have learned to spread out in order to somewhat avoid each other. – Clearly, this depends on how much hikers really want to be alone, which is a parameter that needs to be estimated from real-world surveys. The simulation can then show the results of changes in that parameter, or it can show how much the satisfaction of hikers with respect to that parameter can change when the environment is changed.

2.3. Initial conditions

The learning process needs to be started somehow. For this, an initial population needs to be generated, and they need to compute initial activities and routes. For the generation of the initial population, a **synthetic population generation** module is used. This could use aggregated information, such as known age or gender distribution of tourists, and generate individual agents from this, which have individual age, gender, income, etc. At this point, the implemented module generates agents with random attributes (varying walking speed, different weights for the generalized cost function in the router).

Initial activity plans are constructed from some global knowledge of the area (i.e. agents are told where attractive view points or where restaurants are); initial routes are constructed based on geometrical distance.

2.4. Helper Modules

For computing a solution, the following modules are not needed. However, a simulation system without would be useless:

Recorder/Player. This module receives and processes agents positions and other data from the simulation. It can store them in a file and/or forward them to any viewers which are currently connected to it.

There are several reasons that the simulation does not directly write to log files itself:

- The simulation can be parallelized, and we need the log output of all the instances in a single file.
- Writing to a remote file system (i.e. NFS) can be very slow.
- There is a single interface for viewers.
- By splitting the logging functionality from the simulation, we are able to change implementation details for all of the simulation modules without needing to modify the logging code. For example, one could use a database to log the events instead of a file.

In order to visualize simulation runs, we have developed **Viewer Modules**, which are stand-alone applications that connect to the simulation system via the network. Viewers are built so that they directly plug into the live system. The simulation sends agents' positions to the viewer, which allows to look at the scenario from a bird's eye view and observe how the agents move. It is also possible to send that same data stream to a recorder which records it to file, while a player can read the file and send the data stream to the viewer exactly the same way it would come from the simulation directly. There can be any number of viewers connected to a player module, each showing the same scenario from a different perspective, or displaying different accumulations of events. Finally, in order to deal with data conversion issues, it is also possible to pipe the data stream from the simulation through the recorder directly to the player and from there to the viewer.

We have implemented two different viewers. One displays a 2D view, and is suited for situations where a lot of detailed information is needed, for example while debugging (Fig. 2 and Fig. 3, right side). Also, a 3D viewer has been implemented (Fig. 3, left), as one of our overall project goals is to integrate decisions based on visual stimuli. The 3D viewer connects to the simulation using the same protocol as the 2D viewer. The user can move independently of the agents or can attach the camera viewpoint to a specific agent and see the landscape through the eyes of the agent.

3. Technical Implementation

As described above, the modules need to communicate with each other. The mobility simulation receives plans and outputs events. The mental modules receive events, and output and receive plans. The helper modules normally deal with events only.

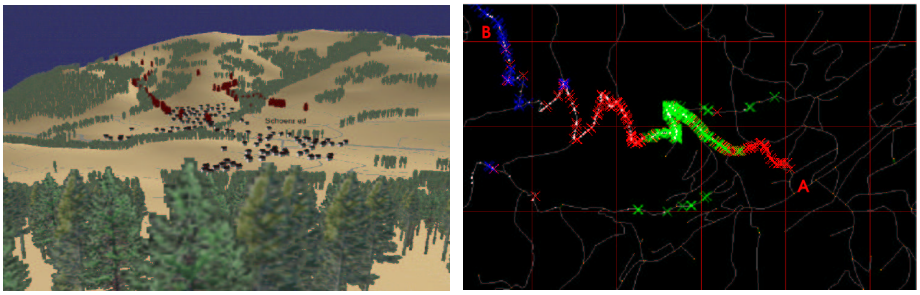

FIGURE 3. A 3-dimensional viewer has been implemented, since one of our overall project goals is to integrate decisions based on visual stimuli (left). Both pictures show agents hiking from a hotel (A) to the peak of a mountain (B). The 2-dimensional viewer (right) displays the events, which contain the agents' perception and performance information, in different colors.

3.1. MPI/PVM

When a *single module is distributed* across multiple computational nodes, one often uses MPI (Message Passing Interface [12]) or PVM (Parallel Virtual Machine [14]). It is also possible to use MPI or PVM for the communication between *different modules* as described in this paper. That approach has, however, the disadvantage that one is bound to the relatively inflexible options that MPI offers. For example, options to add or remove modules during runtime have only recently been added to the MPI standard, and multicast (see below) is not possible at all.

3.2. TCP

On clusters of workstations, MPI is often implemented on top of TCP (Transmission Control Protocol). TCP is the connection based, reliable protocol of the TCP/IP suite. Initially, a connection from the sender to the receiver must be opened. With this connection, both sides can send their messages as the connection is symmetric. TCP guarantees that the messages arrive, in correct order, and without errors.

3.3. UDP

UDP offers, in comparison to TCP, no control for packet loss. This means that there is no guarantee that the sent packets will arrive. The advantage is that there is considerably less overhead. Also, as will be described later, this offers (via multicast) the option to send events only once, even when many modules listen to them. In a TCP-based implementation, events need to be sent to each listener separately. A message that arrives is guaranteed to be error free, since the UDP protocol includes a checksum.

We use UDP to transmit the agent positions to the visualizers. If the network is down for a few seconds, the simulation does not need to slow down because of lost packets. Once the viewer is back on line, it will receive the latest positions.

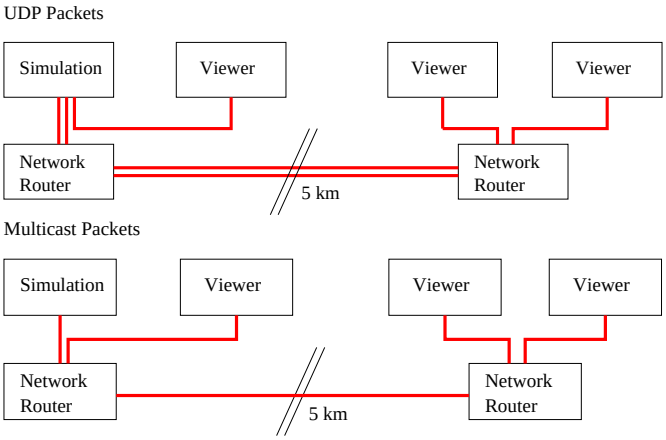

FIGURE 4. Multicast uses one data connection, even if there are multiple receivers subscribed to the multicast channel.

There are other situations in which one may accept the loss of messages. For example, if an agent reports that it is blocked in unexpected congestion (e.g. waiting for a cablecar, traffic jam), it needs a new route instantly. If its request is lost or delayed, it makes no sense for the system to buffer its request, since the agent has moved on, and the location in the original request might now be invalid. A new route computed based on the old information will be invalid as well. It is the agent's responsibility to restate its position again if it does not receive a new route after a certain time has elapsed [7].

The amount of packet loss is strongly dependent on the overall number of packets in the network. In state-of-the-art networks, which today are often 1 Gbit Ethernet, there is hardly any packet loss in the network itself. Losses occur mainly in the sending and receiving network interface cards (NICs), due to overflowing buffers. This is the case, for example, if the CPU is busy so that it cannot read the packets from the buffer quickly enough. The more packets that are sent, the higher the chance that one is lost. With Gbit Ethernet communication, up to 180'000 raw data packets can be sent per second without any losses. Using a naive approach, which is packing one event into one network packet, one obtains 180'000 events per wall-clock second. Since the mobility simulation runs more than 100 times faster than wall-clock time, this results in 1'800 events per simulated second. However, this is not much for a simulation of 1000 or more agents.

Recently, new networking infrastructures have been developed which allow to send packets reliably with UDP. An example is the TNet Hardware [17], used in computer clusters. The TNet hardware assigns a 16 bit CRC (cyclic redundancy check) to each packet. After every transmission over a network link the packet is checked for correctness, and retransmitted if an error is detected.

3.4. Multicasting

Often there is a need for sending the same packet to more than one receiver. This can be achieved by opening multiple TCP connections or by sending multiple UDP packets to the receivers. However, on large simulations, the network interface card (NIC) of the sending host quickly becomes the bottleneck, as it is unable to send out enough packets to keep the receivers fully occupied.

On Internetworks, it is possible to use *multicasting* to send a single message from one computer to several other computers.Because multiple machines can receive the same packet, bandwidth is conserved. Multicasting is particularly useful for any kind of streaming data such as radio or television broadcasts over the network. Its advantage is that the multiplication of the packets for multiple receivers is not done by the NIC, but by the network itself. This allows to avoid the NIC bottleneck.

Multicasting provides groups of hosts, that are referenced using special IP addresses (224.0.0.0 – 239.255.255.255). The sender chooses one of these groups and sends a single packet to this IP address. A receiver must explicitly join a group first, telling its NIC and the operating system to listen for packets sent to this group. An advantage of this addressing scheme is that the sender does not need to know the IP address of the receiver. This simplifies the configuration of the system substantially. The Internet routers ensure that the packets find their way from the sender to the receivers, once they are registered to the multicast group.

A drawback with multicasting is that it has, similar to UDP, no arrival control. There is no feedback to the sender if all packets arrived at all destinations.However, as described before, this is not a problem for data sent to our viewers.

Our project is a collaboration between two institutes at ETH Zürich. One of them is located more than 5 km away from where our computational cluster is. For every viewer that is connected to the simulation, extra bandwidth is needed. Using multicast, we cannot reduce the bandwidth used for one viewer. But as soon as there are multiple viewers looking at the same general area, the bandwidth remains constant (Fig. 4). Sending agent data to multiple viewers is an instance where multicasting is extremely effective. Using multicasting, we were able to allow multiple viewers at the remote institute without saturating the NIC.

3.5. XML: Extensible ASCII Messages

Viewers are built so that they directly plug into the live system. The simulation sends agents' positions to the viewers, which allows to look at the scenario from a bird's eye view and observe how the agents move. As mentioned before, at this point there are two different viewers, one in 2D and mostly intended for debugging, and on in 3D.

For the 3D viewer, more data has to be sent, since it needs also the *altitude* of an agent. One needs the ability to *add* this value to the data stream in a way that there is no need to change existing viewers.

The Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

The main advantage of XML is that there is no need to enforce a mandatory file format between the sender and the receiver.

Tags (e.g. `event`) and attributes (e.g. `agent="42"`) are not defined in any XML standard *per se*. These tags are introduced by the author of the XML document. This makes XML a perfect choice as a format for data exchange among different modules of a simulation. Whenever a module introduces a new kind of message, there is no need for a change in any of the modules that listens to the message stream—unless of course, a module wants to handle this new information specifically. This is also the case for a attribute introduced additionally. It is, however, not possible to *change* the name of existing tags or attributes.

A typical *position update message* used in our simulation system encoded in a XML fragment looks like this:

```
<event type="position" agent="42"
x="588440.1" y="150281.4" />
```

Frequent changes in messages, something that happens often in research and development, are possible. Since the receiving modules search for "keyword=value" pairs, an additional attribute is simply ignored. As a result, there is no need to adopt existing modules. Further, due to the fact that messages are transmitted in plain text, debugging of communication is possible without further tools. The example message is perfectly understandable by all modules if another attribute is added:

```
<event type="position" time="23"
agent="42" x="588440.1" y="150281.4" />
```

Since it is possible to attach existing XML parsers to any I/O stream or buffer, there is no difference between reading messages from a file or receive messages over the network. We tested XML over UDP and TCP, both versions are very flexible. Using UDP, however, if a packet containing a piece of a message is lost during transmission, the resulting stream is not necessarily still a valid XML document. This problem is circumvented by always sending complete messages in a packet. Recall that packets are guaranteed to be error free. UDP packets arriving with an error are discarded; TCP packets arriving with an error are re-transmitted.

Problems with XML are that (i) searching for "keyword=value" pairs is slow, (ii) if part of a XML stream is lost, the whole XML context may become invalid, and (iii) XML is plain text, so we have to convert binary numbers into ASCII characters and back. To overcome problems (i) and (ii), we have developed a parser that is specialized in parsing a certain subset of the XML standard. This subset consists of simple tags with no nested tags inside:

```
<tag {attr="value"} />
```

Note that all information is inside the attributes, and therefore inside the tag as well. Therefore no nesting of tags is possible. This subset parser is written as a substitute for existing XML parsers, such as Expat [5]. Therefore, the programmer's interface is exactly identical.

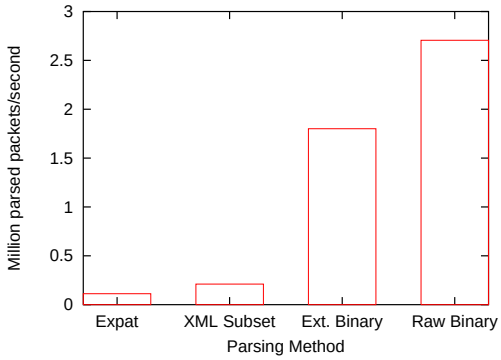

FIGURE 5. Comparison of parsing speed (left): Expat, XML Subset Parser, Extensible Binary Protocol vs. raw binary messages.

We measured the performance of our subset parser by parsing messages of different size for one second. These measurements were done on a 700MHz Pentium III. The messages were taken directly out of the machines main memory, so no network or disk access was involved. The results are that the off-the-shelf Expat parser parses 110'000 messages per second, while our specialized XML subset parser parses 210'000 messages per second (see Fig. 5). Thus, our XML subset parser is faster by a factor of 2.

3.6. Extensible Binary Protocol

The flexibility of XML is achieved at the cost of parsing ASCII files for tags and well-formatted entries. In multi-agent simulations, millions of individual agents are simulated and have to report to external applications such as graphical viewers or storage managers.

Our experiments have shown that the exchange of multi-agent data in the form of XML streams is very flexible but can reduce the bandwidth substantially due to the overhead of parsing the streams and due to converting numbers represented binary internally into ASCII and back.

In terms of CPU time, parsing the same type of streams data all the time is a waste. Often the structure of a message is known once the simulation is running.

A potential strategy is: initially, the consumer module connects to the producer module and receives a description of the available data. Then the consumer specifies which part of the data and which formatting it needs. Based on that specification, the producer begins to output binary streams formatted according to the consumer specification.

For instance, if agents are agents in a cablecar, a 2D viewer will only require x/y coordinates, while a 3D viewer will require x/y/z positions. The whole process can be seen as a user-defined on-the-fly serialization of the producer's objects (i.e. the agents) and their transfer over a high bandwidth binary channel.

We have implemented a protocol that follows these steps:

- A receiving module asks for certain data values to be transmitted in data messages. This is done by specifying the XML tag names, e.g. "agent" or "time".

A example request, as sent over the network, looks like `<request type="event" time="" agent="" x="" y="" />`.

- The sending module transmits a description of future data messages, e.g. `<description type="event" version="1" time="i" agent="i" x="d" y="d" >`. The field `version` is the sequence number of the description. It is unique for each new description sent.
- The sending module from then on sends plain data, packed according to the description into a binary buffer.

As soon as another receiving module asks for additional data values, they are included into the message as well, after a new description is sent to all the receiving modules. A description is sent every second as well, in case a receiving module lost a description or joined the system without requesting a new message format.

Our measurements have shown that this strategy yields in a factor of 17 faster than by using Expat, and in a factor of 8 faster than by using our XML subset parser (Figure 5, 3rd bar).

3.7. Sending raw binary data

For comparison, also the exchange of raw, binary messages was measured. It communicates the same amount of information as the above, but does nothing with it. This results in the 4th bar in Fig. 5. As one can see, the maximum possible speed as measured by this method is only about a third faster than our extensible binary protocol.

3.8. Conclusions

Using the Extensible Binary Protocol, it was possible to visualize scenarios containing more than 1000 agents running more than 100 times faster than real time. An agent number and the x and y co-ordinates of a position update message consume 20 bytes, when packed binary. It is therefore possible to transmit up to 75 agent positions per packet. If the viewer is able to receive 200’000 packets per second, we are able to display 15 million agents per second. Since the viewer uses its host’s CPU cycles for drawing the graphical output as well, the actual number drops again substantially.

4. Discussion and Outlook

It is important to note that the task of the mobility simulation is simply to send out events about what happens; all interpretation is left to the mental modules. In contrast to most other simulations in the area of mobility research, the simulation itself does not perform any kind of data aggregation. For example, link travel times are not aggregated into time bins, but instead link entry and link exit events are communicated every time they happen. If some external module, e.g. the router, wants to construct aggregated link travel times from this information, it is up to that module to perform the necessary aggregation. Other modules, however, may need different information, for example specific progress reports for individual agents, which they can extract from the same stream of events. This would no longer be possible if the simulation had aggregated the link entry/exit information into link travel times.

Despite this clean separation – the mobility simulation and the modules in the physical layer compute "events", all interpretation is left to mental modules – there are conceptual and computational limits to this approach. For example, reporting everything that an agent sees in every given time step would be computationally too slow to be useful. In consequence, some filtering has to take place "at the source" (i.e. in the simulation), which corresponds to some kind of preprocessing similar to what real people's brains do. This is once more related to human intelligence, which is not well understood. However, also once more it is possible to pragmatically make progress. For example, it is possible to report only a random fraction of the objects that the agent "sees". Calibration and validation of these approaches will be interesting future projects.

5. Summary

This paper reports basic elements of a distributed mobility simulation system. The simulation system consists of two layers, the physical layer (mobility simulation), and the mental layer (strategy/plans generation). The mental layer generates plans, which are submitted to the mobility simulation for execution. The mobility simulation returns events to the mental layer. The modules of the mental layer use these events in different ways, for example to score plans, to compute best paths, or to construct mental maps.

Since the communication needs between these modules is substantial, several methods for message passing are evaluated. Traditional approaches, such as MPI or PVM, are too inflexible for what was intended for this project. For that reason, specialized protocols based directly on the operating system are evaluated. These protocols have trade-offs in terms of ease-off-use, bandwidth consumption, and potential message loss. The overall result is that, albeit at the expense of having to use a variety of protocols for different purposes, even with existing technology simulations with thousands of agents running hundreds of times faster than real time are possible with our approach.

6. Acknowledgments

We thank Ingo Opperman for his work on the binary XML-like protocol, and Bryan Raney for the learning mechanism which computed the transition in Figure 2.

References

[1] ALPSIM www page. www.sim.inf.ethz.ch/projects/alpsim/, accessed 2004. Planning with Virtual Alpine Landscapes and Autonomous Agents.

[2] T. Arentze and H. Timmermans. Representing mental maps and cognitive learning in micro-simulation models of activity-travel choice dynamics. In *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, 2003.

[3] A. Babin, M. Florian, L. James-Lefebvre, and H. Spiess. EMME/2: Interactive graphic method for road and transit planning. *Transportation Research Record*, 866:1–9, 1982.

[4] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, (1):269 – 271, 1959.

[5] Expat www page. James Clark's Expat XML parser library. expat.sourceforge.net, accessed 2004.

[6] E. R. Galea, editor. *Pedestrian and Evacuation Dynamics 2003*. Proceedings of the 2nd international conference. CMS Press, University of Greenwich, 2003.

[7] C. Gloor. Modelling of autonomous agents in a realistic road network (in German). Diplomarbeit, Swiss Federal Institute of Technology ETH, Zürich, Switzerland, 2001.

[8] C. Gloor, D. Cavens, E. Lange, K. Nagel, and W. Schmid. A pedestrian simulation for very large scale applications. In A. Koch and P. Mandl, editors, *Multi-Agenten-Systeme in der Geographie*, number 23 in Klagenfurter Geographische Schriften, pages 167–188. 2003.

[9] C. Gloor, L. Mauron, and K. Nagel. A pedestrian simulation for hiking in the Alps. In *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH, 2003. See www.strc.ch.

[10] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, (407):487–490, 2000.

[11] D. Kistler. Mental maps for mobility simulations of agents. Master's thesis, ETH Zurich, 2004.

[12] MPI www page. www-unix.mcs.anl.gov/mpi/, accessed 2004. MPI: Message Passing Interface.

[13] PTV www page. Planung Transport Verkehr. See www.ptv.de, accessed 2004.

[14] PVM www page. www.epm.ornl.gov/pvm/, accessed 2004. PVM: Parallel Virtual Machine.

[15] M. Rickert. *Traffic simulation on distributed memory computers*. PhD thesis, University of Cologne, Cologne, Germany, 1998. See www.zaik.uni-koeln.de/~paper.

[16] P. Salvini and E. Miller. ILUTE: An operational prototype of a comprehensive microsimulation model of urban systems. In *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, 2003.

[17] SCS www page. http://www.scs.ch/references/tnet.html, accessed 2004. The T-Net Hardware.

[18] H. Timmermans. The saga of integrated land use-transport modeling: How many more dreams before we wake up? In *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, 2003.

[19] TRANSIMS www page. TRansportation ANalysis and SIMulation System. transims.tsasa.lanl.gov, accessed 2004. Los Alamos National Laboratory, Los Alamos, NM.

[20] P. Waddell, A. Borning, M. Noth, N. Freier, M. Becke, and G. Ulfarsson. Microsimulation of urban development and location choices: Design and implementation of UrbanSim. *Networks and Spatial Economics*, 3(1):43–67, 2003.

Swiss Federal Institute of Technology, Institute of Computational Science,, CH-8092 Zürich, Switzerland, chgloor@inf.ethz.ch

Swiss Federal Institute of Technology, Institute for Spatial and Landscape Planning, CH-8093 Zürich, Switzerland, cavens@nsl.ethz.ch

Technical University Berlin, Transport Systems Planning and Transport Telematics, D-10587 Berlin, Germany, nagel@vsp.tu-berlin.de,