PART III

Understanding MATSim

CHAPTER 46

Some History of MATSim

Kai Nagel and Kay W. Axhausen

46.1 Scientific Sources of MATSim

As sketched earlier (Section 1.1), MATSim derives from the following research streams:

Microscopic Modeling of Traffic Microscopic modeling was a basis for traffic flow theory from the start (e.g., Herman et al., 1959; Seddon, 1972; Wiedemann, 1974), but the work was limited to individual links, or small sequences of links and could thus not address equilibrium, as aggregate assignment models could from the 1970's onward (see Sheffi, 1985; Ortúzar and Willumsen, 2011). The expansion to whole and large networks came with the increasingly powerful computers in the 1980's, as well as fast and sufficiently accurate flow models (e.g., Schwerdtfeger, 1984; Nagel and Schreckenberg, 1992; Daganzo, 1994; Gawron, 1998).

Computational Physics For MATSim, this development was aided by insights from computational physics, which often adopts simple and very fast models of physical processes and has performed simulations with 10^8, and more, particles since the 1980's (for a contemporary review see Beazley et al., 1995). It was thus clear from the beginning that urban or regional systems with 10^7 or 10^8 persons or vehicles could be simulated microscopically; the research then focused on where necessary compromises would have to be made.

Microscopic Behavioral Modeling of Demand/Agent-Based Modeling According to Russel and Norvig (2010, p. 53), an agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators". In that sense, both the models of Seddon (1972) and of Wiedemann (1974) can be classified as agent-based; this holds even for the simple cellular automata models of Nagel and Schreckenberg (1992), since here driver-vehicle units perceive the distance to the vehicle ahead and act by adjusting their velocity.

Agent-based behavior can also be found at the demand modeling level, where aggregate models, such as the gravity model (Wilson, 1971), can be replaced by person-centric formulations.

How to cite this book chapter:
Nagel, K and Axhausen, K W. 2016. Some History of MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 307–314. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.46.

In that sense, agent-based modeling of travel demand had been developed in Germany since the 1970's (see the references in Axhausen and Herz, 1989), as well as in English-speaking countries, as described in Jones et al. (1983)'s seminal book. While anglophone authors focused on sample enumeration methods to estimate total demand with their activity-based demand models (see Bradley and Bowman (2006) for North American, mostly discrete choice, model-based, developments and Arentze and Timmermans (2000) for an alternative Dutch approach), the simpler German approach was linked to an integral mesoscopic traffic flow simulation in Axhausen (1989), but not used for equilibrium search. It already had, however, a simple description of daily schedule total utility.

Complex Adaptive Systems/Co-Evolutionary Algorithms Nash-equilibrium-like approaches had been developed in transport assignment since the formative Wardrop (1952) paper. These aggregate, flow-based approaches were expanded to account for user perception errors and the social optimum (see Daganzo and Sheffi, 1977). In the late 1990's, transport science addressed the process of learning within the context and new possibilities of "intelligent transport systems", using various smoothing techniques to integrate data from iteration to iteration, reflecting the field tradition. Examples include Chang and Mahmassani (1989); Kaufman et al. (1991); Hatcher and Mahmassani (1992); Smith et al. (1995); Axhausen et al. (1995); Nagel (1995, 1996); Gawron (1998); Mahmassani and Liu (1999); Polak and Oladeine (2002); Arentze and Timmermans (2004). These approaches translated Nash equilibrium logic into co-evolutionary search schemes, which efficiently identified the optima of each agent's daily schedule.

46.2 Stages of Development

46.2.1 Kai Nagel's Perspective

46.2.1.1 Fast Microscopic Modeling of Traffic Flow (University of Cologne/Los Alamos National Laboratory)

Kai Nagel originally wanted to do his PhD (Philosophiae Doctor – Doctor of Philosophy) in meteorology. When funding did not come through, he began exploring alternatives and applied for a position in insurance modeling with Prof. A. Bachem at the University of Cologne. Instead, he was offered a position in operations research, solving problems like dynamic vehicle routing with time windows.

Having some background in computational statistical physics, he soon became skeptical whether it made sense optimizing up to the last second of a time window, while simultaneously facing a highly stochastic transport system. Using his training, he embarked on building a microscopic model of the transport system, in particular single-lane (Nagel and Schreckenberg, 1992; Nagel, 1999) road traffic on long links, as well as combining such links to large-scale network-based simulations, where each vehicle follows its own individual route (Nagel, 1996), including adaptive dynamics, being influenced most heavily by Arthur (1994). That paper already (Nagel, 1996) describes what is still the main MATSim architecture, where agents have many different plans, keep trying them out and eventually settle on the best option. In contrast to the current approach, in that paper, all plans were pre-computed; i.e., there was no innovation during iterations. This was possible because the network was much coarser than what we use today, making pre-computing route plans with enough diversity easy.

46.2.1.2 TRANSIMS (Los Alamos National Laboratory/Santa Fe Institute)

Some of the above PhD work was done during Kai's tenure as a Graduate Research Assistant at LANL (Los Alamos National Laboratory). After his PhD, he moved to LANL, where he worked

with the TRANSIMS (see, e.g., Smith et al., 1995) team, under the leadership of Chris Barrett. The TRANSIMS project used some of the design described above, most notably the cellular automata approach to road traffic modeling, which was thus extended to multi-lane traffic (Nagel et al., 1998), to intersections (Nagel et al., 1997) and to massive parallel computing (Nagel and Rickert, 2001).

In terms of software design, TRANSIMS was a collection of stand-alone modules, coupled by a script. For example, the population synthesizer would generate a population file, the activity generator would take the population file as input and generate an activities file as output, etc. Iterations were done by running the traffic microsimulation (called mobsim in MATSim) based on plans and outputting average link travel times and then running the router based on link travel times and outputting plans.

46.2.1.3 MATSim in C++ (ETH Zürich Computer Science)

Kai Nagel moved to ETH Zürich Computer Science in 1999. It was difficult there to continue with TRANSIMS, partly because TRANSIMS was not under an open-source license at that time and also because TRANSIMS fell under U.S. technology export restrictions for some time. As a result, MATSim was started.

MATSim was different from TRANSIMS from the beginning in two important ways: (1) it tried to be more lightweight, i.e., running much faster, specifically by using the queue model (Gawron, 1998), rather than the cellular automata model for network loading and (2) other than TRANSIMS, agent properties such as demographic data, activity patterns or routes were no longer distributed across multiple files, but contained in one hierarchical XML file.

Another difference later appeared, which went back to the Nagel (1996) approach, but this time really followed Arthur (1994) by giving each individual agent its own memory (Raney and Nagel, 2006). After experimentation with relational databases such as MySQL (MySQL, accessed 2014) or Oracle (ORACLE www page, accessed 2005), it was eventually decided to implement MATSim as an object-oriented database in memory, i.e., by first reading in all XML files, modifying the data in computer memory RAM during a run lifetime and writing the data back from memory to XML files at the end of the run. The decision was based on the observation that the MATSim data model was described much better by XML files and that conversion to the relational format was impractical, prone to errors, and too slow if not kept in memory during iterations.

46.2.1.4 MATSim in Java (TU Berlin Transport Engineering)

Michael Balmer wrote his dissertation at ETH (see below) about demand modeling for MATSim, i.e., about the upstream process that leads to initial plans (Balmer, 2007). That project, different from the main MATSim code at that time, was written in Java. Along with the assessment that Java would be the better language than C++ to continue development, it was decided to use Michael Balmer's code as starting point for a Java version. Arguments for Java included:

- Java is more restrictive. For example, in Java, objects are always passed by reference, [1] while in C++, one has the choice between passing a pointer, a reference, or a deep copy of the object. Since standards are difficult to enforce in academic environments, a more restrictive language seemed (and still seems) the better choice.
- Java runs well on many platforms. This allowed (and still allows) us to let people work on their favorite platforms, be it Linux, Microsoft Windows, or Mac.
- There is good non-commercial support for Java; for example, the Eclipse IDE and numerous powerful libraries.

[1] We abstract from the notion that Java "passes object references by value".

- The Java compiler is easier to handle. For example, there is no extraction of header files and the Java compiler sorts out, by itself, the sequence in which modules need to be compiled.
- For our applications, Java was consistently *not* slower than C++. This assessment was based on several years of teaching a MATSim class at ETH Zürich, where computer science students implemented simple versions of MATSim in a programming language of their choice. Typically, while the fastest C++ code may have been 30 % faster than the fastest Java code, the slowest C++ code normally was a factor of 3 slower than the slowest Java code. In other words, while C++ gives more opportunities for optimization, it also gives more opportunities for very serious performance degradation. This assessment is corroborated somewhat in the literature (Prechelt, 1999), where, in one example, it is demonstrated that interpersonal differences within the same language are of the same magnitude as differences between languages.

  In addition, it seems that the gap between C++ and Java has narrowed further since then. Important differences remain in numerical applications, also partly because C++, other than the Java, allows operator overloading.[2] However, MATSim's agent-based approach means that complex objects are handled much more frequently than true numerical computations.
- One reason for using C++ was that it could be combined with MPI, which is a reliable message passing standard for parallel computing. Parallel computing was necessary both for performance reasons and to be able to run simulations that needed more than about 4 GB of memory—the maximum that could be addressed with the 32 bit architecture standard at that time. MPI is also available for Java, but it is much less well maintained.

  With the advent of the 64 bit architectures, the second reason for parallel computing became obsolete. In addition, with Kai Nagel now at a transport engineering department, it seemed that making conceptual progress was more important than keeping the parallel computing edge, especially since the maintenance of parallel code *permanently* consumes additional resources.

  With the decision to give up on parallel computing, it was no longer necessary to maintain compatibility with MPI; thus, the move to Java was facilitated.

In terms of language, C# might have been an alternative to Java. However, C# depends much more on the Microsoft Windows platform, and community support is not as good as it is for Java.

Clearly, the code by Michael Balmer already had all the necessary data classes, readers and writers. The code was used as a starting point to re-implement MATSim in Java. Nevertheless, many important elements like mobsim, events architecture, scoring, routing, and co-evolutionary architecture had to be re-implemented. It took about two years from making that decision to the first plausible run of MATSim in Java.

Important early steps with MATSim in Java were to add time choice (Balmer et al., 2005b) and mode choice (Rieser et al., 2009) as additional choice dimensions beyond route choice. A summary of the status around 2008 was written by Balmer et al. (2009b).

46.2.1.5 Code Reorganization

The C++ version of MATSim was, similar to the original TRANSIMS, a collection of stand-alone executables coupled by scripts. For example, the router would read plans and events and replace some of the plans by other plans with modified routes. The program flow was organized with shell scripts and makefiles. Later, it was possible to start all modules simultaneously where they used messages to interact (also see Gloor and Nagel, 2005), but the file-based and scripted interaction always remained available.

[2] See `http://en.wikipedia.org/wiki/Operator_overloading`.

That approach had, in consequence, very clearly defined interfaces, i.e., the files. Exchanging information not included in the files meant changing the readers and writers on *both* sides, which was, in consequence, rarely done; stand-alone modules instead tried to work with the information they had.

When MATSim was re-implemented in Java around 2006/07, it was re-implemented as one system. Now, everything could interact with everything. For example, a router could modify the network, compute routes on the modified network and then modify it back. Clearly, it could make an error in the process, thus erroneously modifying the network. In this way, any module could modify any data of MATSim, greatly increasing the scope for misunderstandings and errors.

What created even more problems, however, were extensions to the program flow. The program flow was, as it still is, organized by the `Controler` class.[3] Originally, everybody who wanted to change the program flow and insert his or her own research modules, would inherit from `Controler`, override some methods and insert his or her own instructions. This however, meant that it was impossible to *combine* the extensions without possibly massive manual interventions, illustrated as follows.

For example, assume the core program flow as

```
class Controler {
   void run() {
      ...
      aMethod() ;
      ...
   }
   void aMethod() {
      doA() ;
      doB() ;
   }
}
```

Also assume an extension called `MyControler` from one researcher and another extension called `YourControler` by another researcher:

```
class MyControler extends Controler {
   @Override
   aMethod() {
      doA() ;
      doMyStuff() ;
      doB() ;
   }
}
```

```
class YourControler extends Controler {
   @Override
   aMethod() {
      doA() ;
      doYourStuff() ;
      doB() ;
   }
}
```

If you wanted to combine both approaches, you could neither say `YourControler extends MyControler` nor `MyControler extends YourControler`, since either way one of the two extensions would get lost. In this simple case, one could possibly address the problem through manual

[3] Mis-spelled since its inception.

intervention, but in more complicated situations this would no longer possible without extensive additional testing.

Therefore, in 2008, a decision was made to make MATSim more modular. The first step in that direction was a decision to submit the whole MATSim repository to frequent refactorings, i.e., to *not* leave the code alone as much as possible, instead forcing the community to get used to frequent changes of code, while maintaining functionality. To facilitate that approach, coverage by automatic regression tests on the build server was hugely increased and all developers were encouraged to write automatic regression tests for their own code and projects.

The changes since then are too numerous to be listed here. They include, in particular, fairly restrictive data classes no longer extended or modified by every scientific project, and well-defined extension points in both the iterative loop and inside the mobsim. See Chapter 45 for currently existing extension points.

46.2.2 Kay W. Axhausen's Perspective

46.2.2.1 ORIENT/RV: Parking in Travel Demand Models (Karlsruhe University)

In 1984, Kay Axhausen returned to Karlsruhe University[4] after two years doing an MSc degree at the University of Wisconsin, to start his PhD (Philosophiae Doctor – Doctor of Philosophy) at the IfV (Institut für Verkehrswesen/Institute for Transport Studies). At that time, the IfV already had a long tradition of traffic flow analysis (Leutzbach, 1972) and agent-based traffic flow simulation, as pioneered by Wiedemann (1974) (see also Leutzbach and Wiedemann, 1986). In this environment, Sparmann and Leutzbach (1980) had implemented a sample enumeration-based simulation of traffic demand in the spirit of Poeck and Zumkeller (1978). This approach took the daily schedule of the traveler and simulated it activity-by-activity, including the necessary travel. Neither the traffic flow nor travel demand simulations aimed for equilibrium, but, in line with discussions at the time, both were more interested in the underlying behaviors (e.g., Jones et al., 1983).

Faced with a project to simulate parking as an extension of Sparmann's ORIENT approach, it became clear to Axhausen that sample enumeration approaches could not account for the temporal and spatial competition for parking spaces, but that the event-oriented approaches of the traffic flow model naturally could. Merging the two approaches was the natural solution and he then designed it for ORIENT/RV (Axhausen, 1989). Given the need to model the flow of traffic on the roads as part of the daily dynamics, the approach of Schwerdtfeger, an IfV colleague, was a natural and computationally-efficient choice. Schwerdtfeger (1984) had developed a mesoscopic simulation of traffic flow, which retained the agent-resolution, but employed macroscopic link-performance functions to calculate link speeds.

The work of Swiderski (1983), a second IfV colleague, started Axhausen thinking about the need to account for the constraints imposed by travelers' mental maps. As a full implementation of a mental map is impossible, even with today's computers, he chose to condition travelers' route choices on their travel time expectations, which were based on shortest-paths over an initially empty network. The agents reconsidered their routes at every junction if the experienced travel time deviated beyond an adaptive threshold from expected travel times. In this case, the route was recalculated with the current speeds. The framework was used to iterate (Axhausen, 1990) the expectations via shortest-paths based on stored mean travel times from the last iteration, but no formal tests of equilibrium were conducted, nor was the number of iterations extensive.

In the MATSim context, the competition for facilities was taken up by Horni et al. (2009). Reconsidering routing decisions while already being en-route was taken up by Dobler (2013),

[4] Now: Karlsruhe Institute of Technology (KIT).

Number and type of activities
Sequence of activities

- Start and duration of activity
- Composition of the group undertaking the activity
- Expenditure division
- Location of the activity
    - Movement between the sequential locations
        - Location of access and egress from the mean of transport
        - Parking type
        - Vehicle/means of transport
        - Route/service
        - Group traveling
        - Expenditure division

Source: Axhausen (2014, 2006, 2009)

Figure 46.1: Behavioral dimensions to be included in a fuller scheduling model.

where he showed that such an approach can approximate the equilibrium in a small number of iterations.

46.2.2.2 From EUROTOPP to MATSim (Karlsruhe, Oxford, London, Innsbruck, Zürich)

The first framework program of the European Union offered a chance to continue with the work in a larger context; unfortunately, this extended version of ORIENT/RV never went beyond the design stage (Axhausen and Goodwin, 1991). The EUROTOPP approach was later implemented in a changed form at the IfV, again by Zumkeller, who also had been one of the partners of the first framework project (Schnittger and Zumkeller, 2004), and his students.

Moving to Oxford, London, Innsbruck and then Zürich in rapid succession kept Axhausen from initiating serious work on a large-scale simulation system. The focus switched to data collection and choice modeling and collaboration on travel demand simulation with Kai Nagel began when he also joined ETH in 1999. While this was initially low key, Michael Balmer and David Charypar's move to Kay Axhausen's group after Kai Nagel's departure to TU Berlin jump-started further work, which is now documented in this book.

46.2.2.3 "Best Response" and Further Choice Dimensions (ETH Zürich Transport Engineering)

Departure time, mode and route choice are the heart of the transport modeling enterprise and were addressed in MATSim almost from the start (Raney and Nagel, 2004; Balmer et al., 2005b; Rieser et al., 2009). Work in Zürich addressed further behavioral dimensions, as shown in Figure 46.1. earch or past studies, which did not produce stable enough code for general use. It is clear that there are more dimensions to consider. Those listed in the figure are only the more obvious examples: for example, rail travel service class or activity engagement intensity are not addressed .

Today, MATSim takes the activity chain and schedule, as given from the initial demand generation process, as input; modern "activity based-models" make it sensitive to accessibility, understood

as the logsum term of the included destination and mode choice model (route choice is generally excluded in those models) (see Ben-Akiva et al., 1996, for an early example).Computational overhead costs of calculating non-chosen alternatives sets becomes prohibitive at the scale for which MATSim is designed, so alternative approaches were explored. Meister developed a **genetic algorithm on a household-basis to find optimal schedules for all members simultaneously** (reported in Meister et al., 2005), but only its time-of-day choice element was used in later scenarios (Meister et al., 2006). Feil set about finding a **best-response, but computationally fast approach to the optimization of the number and sequence of activities into a schedule** (Feil, 2010). While he made substantial progress using a tabu search and a cloning approach, it is still too slow as it currently stands. Fourie's PSim (see Chapter 39) might remove that constraint.

While Meister and of Feil's approaches, as well as the standard MATSim routing algorithm, attempt to directly provide best response solutions, the standard MATSim evolutionary algorithm also moves in the direction of good or best response (also see Section 97.3.1). With these approaches, it is impossible to directly model **destination choice**, since the best response destination would just be the closest possible destination (Horni et al., 2009). The problem: destinations similar from the analyst's point of view are quite different from each person's point of view: for example, allowing different types of leisure activity. As further explained in Chapter 27, the problem was addressed by attaching randomness directly to each person-alternative-pair (also see Horni et al., 2012b).

The need to address **parking** is obvious and even more so when considering electric vehicles and their current need to be recharged during the course of a day. Waraich addressed both aspects by integrating a local search into the overall MATSim iteration scheme to identify preferred parking spaces near the final destination (Chapter 13). Dobler's approach (Dobler, 2013) to evacuation is similar, but does not iterate, since that is not relevant for evacuation modeling. Waraich's local search can be extended with personalized walking time values.

The **group composition for joint travel and joint activities** is essential for making progress on a number of fronts, but especially to understand destination choice and activity generation. Gliebe and Koppelman (2005) or Zhang et al. (2005), for example, have proposed discrete choice models for household activity allocation. However, these approaches cannot be easily integrated into MATSim because of their computational costs. They are also too restrictive, with their exclusive focus on the household. Based on parallel empirical work on social networks (see Larsen et al., 2006; Kowald et al., 2013), Dubernet is currently exploring new game theoretic approaches to co-ordinate the timings and activities of households and wider social networks. These social networks are generated using the approach of Arentze et al. (2013), which was estimated against Swiss data for leisure social contact (Kowald and Axhausen, 2012) so as to reproduce measured characteristics of the real network, such as homophily, clustering and average number of leisure social contacts.

The **expenditure division** question is a promising research avenue (Section 97.6) not yet explored by transport planning and clearly interacting with joint activity participation and travel.