

# Using parameter optimization to calibrate a model of user interaction

Tommy Baumann\*, Bernd Pfitzinger<sup>‡§</sup>, Dragan Macos<sup>†</sup>, Thomas Jestaedt<sup>‡</sup>

\*Andato GmbH & Co. KG, Ehrenbergstraße 11, 98693 Ilmenau, Germany. tommy.baumann@andato.com

<sup>‡</sup>Toll Collect GmbH, Linkstraße 4, 10785 Berlin, Germany. {bernd.pfitzinger|thomas.jestaedt}@toll-collect.de

<sup>§</sup>FOM Hochschule für Oekonomie & Management, Zeltnerstraße 19, 90443 Nürnberg, Germany.

<sup>†</sup>Beuth Hochschule für Technik Berlin, Luxemburger Str. 10, 13353 Berlin, Germany. dmacos@beuth-hochschule.de

**Abstract**—Simulation models of real-world distributed systems depend both on the accuracy of the underlying model and the interaction between user and system. The user interaction is typically modeled as stochastic process depending on parameters and distributions describing the actual usage. Accurate data is often not available and (manual) assumptions are necessary. Taking an existing large-scale simulation model of the German tolling system we discuss the use of a genetic optimization algorithm for calibrating the simulation model.

## I. INTRODUCTION

**D**ISTRIBUTED software-intensive systems become a part of everyday life. The engineering and operations of these systems is not yet well established: Most techniques in use focus on standalone systems [1] and even there successful implementations are not guaranteed [2]. Instead of the engineering aspects one can rather argue [3] that the integration of subsystems into a functioning system-of-systems becomes a core strategic business capability. Whether it is the engineering, integration or the subsequent operation of a highly automated software-intensive system – many dynamic aspects depend on its usage and often unknown user behavior.

Taking the example of the German automatic toll system for heavy goods vehicles (HGVs) – a typical example of a modern toll system based on global navigation satellite systems (GNSS) [4] – we complement the system operations and system design through simulations [5]. Having a detailed, realistic simulation model at hand it is possible to predict the upcoming operational behavior (e.g. for fleet-wide updates) even for systems under design e.g. when a different system architecture is proposed. In both scenarios simulation results yield numerical results to support decisions and to reduce the risk inherent in any software development process. Particularly in the latter case simulation models help to explore different solutions and to create exact specifications right from the start of a software development project.

“Good models are essential for communication among project teams and to assure architectural soundness” [6]. Yet the emphasis on communication (even in more formal methods as UML [6]) creates a source for misunderstandings and errors through the inexact verbalization of the requirements. To reduce the ambiguity we use executable models, i.e. implement the requirements in a model that can be compiled and executed [7]. From the very beginning this allows verifying

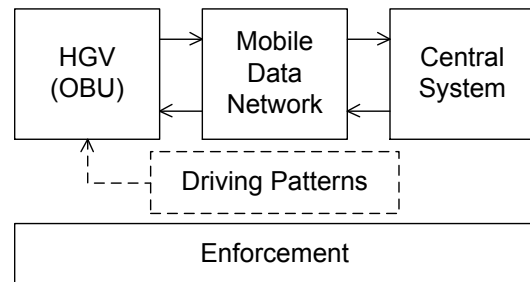


Fig. 1. High-level system design of a GNSS-based electronic tolling system and its dependency on the user interaction (driving patterns).

the system requirements by comparing the simulation results with the expected behavior. As the development of the system progresses the simulation models can progress as well (to give an accurate model at the level of abstraction available at that time) or remain at a reasonable level of detail sufficient e.g. to simulate the overall system dynamics. In the case of the German toll system we have established a simulation model of the automatic tolling process and used it both to predict the operational behavior of the existing system [8] and to aid the software development process to better scope proposed changes (e.g. [5]).

In the next section we introduce the simulation model of the German toll system. To get an executable system we need two models: A model of the technical system and a model of the user interaction. At present not much is known about the user interaction (due to technical restrictions and data protection regulation) and we started with a simple model that can be parametrized to fit the observed dynamic behavior of the toll system. Section IV uses a genetic algorithm (introduced as a separate model in section III) to find a set of parameters that best reproduces the actual system dynamics. The initial results are given in section V followed by a brief summary.

## II. SIMULATION MODEL OF THE GERMAN TOLL SYSTEM

Starting with an existing model of the German toll system and a simple model of the user interaction [8] we take data observed in the real-world system to measure the accuracy of the simulation model. This section gives a brief overview of two models necessary to reproduce the dynamic behavior of the German toll system. In addition we need to address the

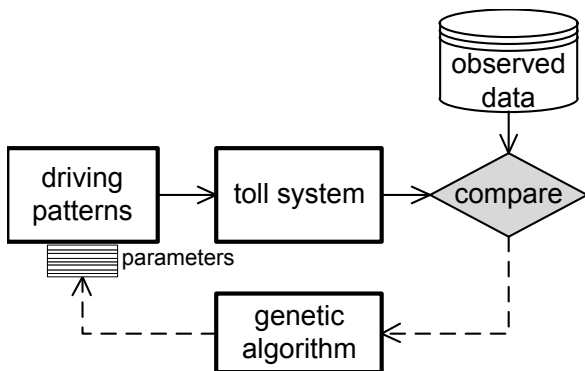


Fig. 2. The existing models for the driving patterns and the toll system are complemented by a dataset of the known dynamic behavior of the toll system. Adding a fitness functions allows comparing the simulation results with the observed data and the use of a genetic algorithm (again as a model) to choose the right parametrization.

term “accuracy” to define the appropriate in our context more clearly.

#### A. System model

The main model from the perspective of system operations or engineering is of course the model describing the (technical) toll system (Fig. 2). It is a discrete event simulation model of the German automatic toll system encompassing a fleet of almost 800 000 HGVs, each with an on-board unit and access to the central system via a mobile data network. The major processes at time scales of 1 second and above are included in the model (and some at considerably shorter time scales) including the network connection with their respective bandwidths and latencies (but not modeled on the level of the TCP/IP protocol). Simulating at a scale of 1:1 we do not introduce ambiguities due to scaling (especially since the system under consideration is in parts highly non-linear) but have achieved a high simulation speed. Looking at fleet-wide updates taking more than a month the model itself works with typical time scales of one second. All in all the typical simulation performance after a number of performance improvements [9] gives execution times of less than 10 hours for the simulating the whole fleet over half a year.

The model has been verified through software inspection [10] and validated through the comparison with the data observed in the real-world system. At present the biggest source of uncertainty is introduced by the driving patterns (our model of the user interaction).

#### B. “Driving patterns”: A model of the user interaction

The model of the toll system depends on the external stimulus of the user interaction. With an emphasis on the fleet-wide propagation of updates we started with a simple model describing only the temporal behavior (fig. 3): The points in time when an HGV is powered on (or off) and when a toll event is created. Since we do not yet include any geographic information the toll events correspond to the HGV driving

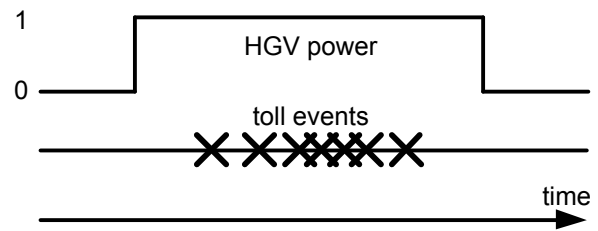


Fig. 3. The user interaction is modeled as the points in time where power-cycles or toll events are registered for a given HGV.

distance of 4.2 km (the average length of a toll segment) at an average speed of  $\approx 80$  km/h.

To achieve a realistic update behavior (taking several weeks to reach the whole fleet) we start with a probability distribution of “active weeks”, i.e. the probability that a given HGV is powered-on (at least once) in  $N$  of  $M$  weeks and take data observed in the actual system and existing HGV-fleets used for testing (typically covering several hundred to a few thousand HGVs). This is in turn followed by probability distributions determining the number of “active days” within a week, the number and duration of power cycles per day and the time during the day when power-cycles take place (for details see [8]). The driving patterns are (manually) calibrated to reproduce the update behavior observed in the real-world system – i.e. looking at time scales of several weeks taking one sample per day.

However, in reality not much is known about the user interaction. Apart from small test fleets no data is available on the power-cycles of the HGVs (even the average speed is not known): Most often the data is not collected and even if data is available data protection regulation often prohibits its use [11]. Looking at the simulation results in more detail it is very difficult to manually adjust the parametrization as to reproduce the intra-day dynamic system behavior (see fig. 4). To make matters worse, some processes are deliberately made strongly non-linear (e.g. to favor updates during the night-time or to protect the central system when it is operating close to the specification limit).

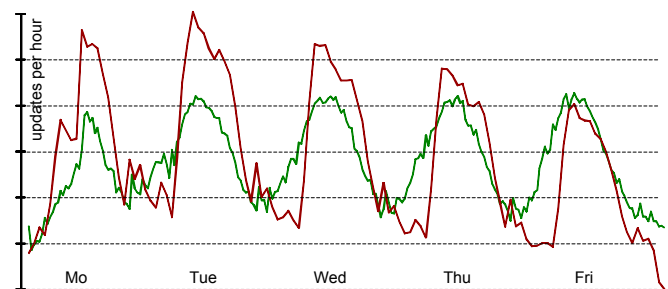


Fig. 4. Example of the simulation results (green line) in comparison to the observed data (red line) of the hourly update rate.

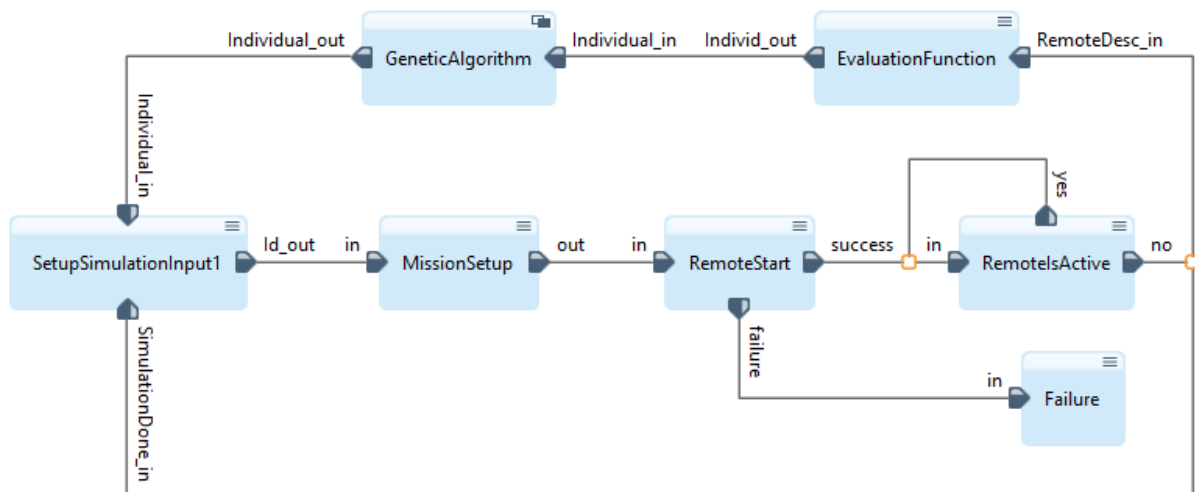


Fig. 5. MSArchitect model using the genetic algorithm to optimize the user interaction (driving patterns) model.

### C. Measuring accuracy

To overcome these difficulties we started to let an optimization algorithm adjust the parameters to improve the simulation results. Keeping in mind the mantra “good enough” [12] we need to look at the context in which the simulation results are used:

- *Updates* take several weeks to propagate across the whole fleet. A comparison of the daily data is sufficient to measure the quality of the simulation results.
- *Utilization* of servers or data networks changes rapidly and a time resolution of an hour or less needs to be taken into account.

With the emphasis on fleet-wide updates we compute the fitness as rms-deviation using one data point per day. Depending on the use case this may include the one or several components that are updated: geo and tariff data and the OBU software. For the purpose of this article we take the OBU software as the only input to the fitness function and compute the rms-deviation only for the first weeks after the start of a software update. Only the first weeks of an update are influenced by the system simulation model (and its parametrization of the update rate), very soon the HGV activity is the limiting factor (i.e. a substantial part of the HGV fleet connects only rarely to the central system either because the HGV is constantly powered off or in a foreign country).

### III. SIMULATION MODEL TO OPTIMIZE THE DRIVING PATTERNS

The limited knowledge of the actual user interaction (as mentioned above) combined with the difficulty of setting the parameters of the user interaction manually necessitates the use of an automatic optimization algorithm to calibrate the model with the data available. Consequently the model of the German toll system is connected to an optimization algorithm (see Fig. 2). To separate these tasks we introduce a second optimization model responsible for the optimization of the

driving patterns (user interaction). Using a genetic algorithm the model iterates automatically over different parametrization improve the simulation results for the model of the German toll system.

To that extent the existing model of the German toll system needed only minor modifications: Any parameters intended for optimization were implemented as explicit parameters at the top-level of the model and exported to the optimization model. The existing models generating the driving patterns and simulating the tolling system are now integrated into one model and followed by a *fitness function* evaluating the deviation between the simulation results and real-world data (typically of the progress of fleet-wide updates computed as the rms of the daily version status).

The idea of using a separate model controlling the optimization process, including structural and parameter modifications as well as evaluation of the model to be optimized, is a basic concept of Simulation-Driven Design [7]: There it is called *Executable System Design Process*, defined as an automated series of design steps, which alter the Executable System Specifications (in our case the model to be optimized) in a formal, consistent, and self-contained manner to enable processing [13]. Three base types of components are differentiated:

- *Execution components* are responsible for the execution of the whole, of the parts or of abstractions of the embedded executable system specification as well as execution of associated systems.
- *Control components* implement the evaluation of constraints, rules and objective functions to control the execution of process components.
- *Generator components* generate, transform and extend executable models to comply with different purposes, abstraction levels, parameterizations and structural architectures.

Looking at the simulation model used for optimizing the driving patterns (Fig. 5) we recognize all three types of compo-

nents: Starting with the block instance `GeneticAlgorithm` we recognize a generator component responsible for generation of individuals with different genomes. The genome represents a parametrization of the toll system in the form of a parameter vector and is sufficient to execute the model of the toll system for the given individual. The value of each parameter is bound to a defined range and granularity (i.e. a bit representation of the value) – at present the parameters are the number of active weeks of domestic and foreign trucks (see section II). In the next step `GeneticAlgorithm` sends the representation of the individual via its output port and connection to the block instance `SetupSimulation` responsible for the preparation of the toll system model. `SetupSimulation` (obviously a generator component) creates the necessary environment e.g. the directory structure for simulation in- and output including the parameters and any necessary configuration files. In addition `SetupSimulation` checks the available resources and chooses the number of simulations to be executed in parallel (depending on the number of CPU cores available and the population size). However, the final setup of a simulation run is delegated to another generator component (`MissionSetup`). It generates a mission descriptor data structure, containing the command line parameters to setup and execute a simulation run. This data structure is in turn sent to the first execution component (`RemoteStart`) to execute the model on a specific CPU core. While the simulation run is ongoing two further execution components (`RemoteIsActive` and `Failure`) observe the simulation run and inform the subsequent block instances `SetupSimulation` and `EvaluationFunction` when the simulation run has finished: `SetupSimulation` prepares a new simulation run if necessary an `EvaluationFunction` evaluates the simulation results with respect to the fitness function (see section IV). Since block instance `EvaluationFunction` decides about continuing the optimization loop or not it is a control component.

#### IV. APPLIED OPTIMIZATION ALGORITHM

To solve our optimization problem we decided to use a genetic algorithm. The straight-forward implementation of a parallelized genetic algorithm was the main reason to choose this optimization algorithm. A genetic algorithm is a search algorithm for optimization purposes based on the mechanics of natural selection and natural genetics. Genetic algorithms are able to avoid getting stuck in a local optimum in the search space, can be used in high-dimensional search spaces and are trivially parallelized (“embarrassingly parallel”, [14]). They belong to the group of so called meta heuristics – search methods for approximate solutions [15].

Fig. 6 gives the generic flow chart of the genetic algorithm used: At the beginning an initial population of a fixed size is generated either randomly or using previously available individuals. To limit the search space we choose the parameters to be within pre-defined intervals. Next the fitness of all individuals in the population is computed (step 2 in Fig. 6) to yield the ‘parent population’. In the third step the algorithm

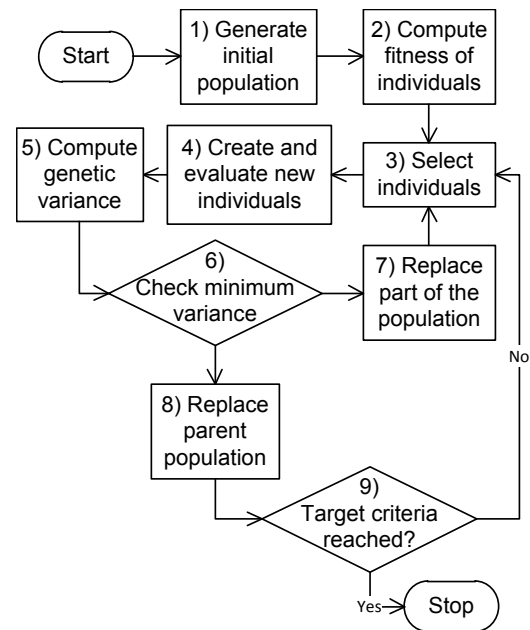


Fig. 6. General sequence of action of the genetic algorithm used (based on [16, 17]).

randomly selects two sets of parents in a *tournament selection* to choose those two parent individuals with the best fitness value.

Once two parent individuals are selected one child individual is created using uniform crossover without mutation (step 4 in Fig. 6). For crossover we select one part of the genome of the first parent individual and the complementary part from the second parent individual. In order to do this a crossover point is chosen randomly. At present we are not using mutation where parts of the genome of an individual are changed randomly to increase population diversity. From the perspective of optimization theory this method is used to overcome local optima [18] – which is implemented in our case by enforcing a minimum variance within the population (step 6 in Fig. 6).

When the child population is fully populated the optimization model starts to evaluate the fitness function by executing simulation runs of the toll system model (as described above). When the evaluation is finished the algorithm checks the genetic variance [19] inherent within the child population (step 5 in Fig. 6). If the variance becomes too small a part of the population is replaced by new randomly generated individuals (step 7 in Fig. 6) otherwise the child population replaces the parent population (step 8 in Fig. 6) and the optimization run continues until the target criteria are met (step 9 in Fig. 6). In addition we use step 6 to check whether the optimization run finds better solutions and again replace part of the population if the results did not improve within 6 generations.

In our case we use fairly small populations with 765 individuals i.e. at a scale of 1:1000 and a genome of 32 parameters each expressing the probability for an HGV of

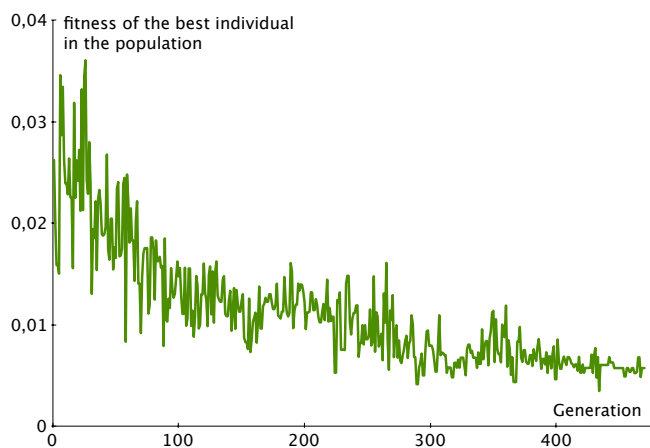


Fig. 7. Fitness of the best individual per generation over the optimization run.

being active (with at least one power cycle) a certain number of weeks within a 15 week period (once for German HGVs and foreign ones). The optimization algorithm starts with a given probability distribution based on historical data from the real-world system (or a previous optimization run). In the next step the fitness function (also called target or evaluation function) is evaluated to quantify the fitness of each individual. Since the evaluation of each individual is independent the algorithm is trivially parallel and we send each computation via the `Remote` component to a different CPU node (see section III) for execution, i.e. the model of the German toll system with the parameterization given by the individual is executed for each individual in parallel.

As an example we take a fleet-wide software update that was rolled out in spring 2012. Using fleet-wide timing parameters the roll-out was configured to spread over 6 weeks where a single update needed less than 10 minutes to download under optimal conditions. To achieve a reasonable number of function evaluations in the optimization we run the simulation model at a scale of 1:1000 resulting in an execution time of less than one minute for the time period of interest – 4 weeks before the start of the update and the first 10 weeks of the update. The simulation model itself is not modified from previous versions [5] and each instance works within its own subdirectory to read and write intermediate results as necessary. At the end of each simulation run the fitness function is evaluated expressing the quality of each individual as the square deviation between the simulated and the real world update roll-out curve (see section V).

## V. OPTIMIZATION RESULTS

For the purpose of this discussion we choose a software update in 2012. Without access to the optimization algorithm the simulation model was parametrized using statistical data from the real-world system and subsequent minor manual adjustments. In comparison we give the results after performing an optimization run with the simulation model at a scale of

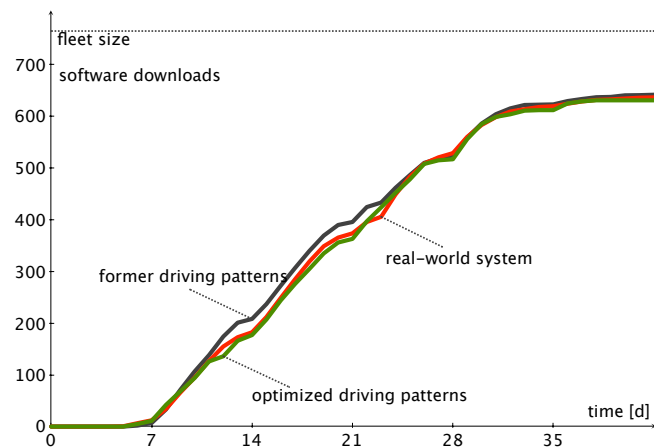


Fig. 8. Simulation results for a fleet-wide software update before and after optimization (red line: data observed in the real-world system, black line before optimization and green line after optimization).

1:1000, a population size of 64 and some 400 generations.

Fig. 7 gives the evolution of the fittest individual per generation. The fitness gradually improves over run-time but from generation to generation it can give worse results since the algorithm creates a completely new population for each generation without keeping the best individual around.

Looking at the results (Fig. 8) the optimized driving patterns perform considerably better during the main update phase. The real-world system is configured to give an almost constant rate of updates during the first weeks (red line in Fig. 8) where the OBUs decide randomly when to download the update according to fleet-wide timing parameters. After a few weeks the update rate is determined mostly by those OBUs that are rarely active within the German mobile data networks and no longer depends on the download parameters. So far the previously existing user interaction model typically produces too many updates during the 2<sup>nd</sup> and 3<sup>rd</sup> week (black line in Fig. 8) even though the model uses statistical data gathered in the real-world system on Toll Collect test fleets.

To emphasize the time period where the algorithms of the toll system determine the download rate rather than rarely visiting HGVs we compute the fitness function only for the initial 6 weeks. This results in a marked improvement of the simulation results (green line in Fig. 8) for the time period shown. However, since the long-term activity pattern was in this case not subjected to optimization the optimized driving patterns give somewhat worse results for longer time periods (not shown). Taking the deviation from the data observed (Fig. 9) the improvement during the first two weeks of the software update are obvious.

Optimizing the probability distribution for the weekly activity pattern quickly improved the simulation results. However, deviations are still visible even when using a coarse time resolution of one day: Typically at the end of the workweek the difference is biggest and changes its sign with the coming week. This suggests that at least further parameters need

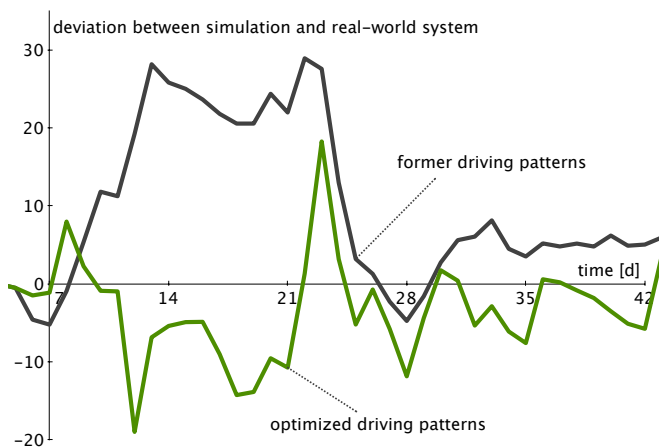


Fig. 9. Difference (in cumulated downloads) between the data observed in the real-world system and both simulation runs (black line before optimization, green line after optimization).

systematic optimization or even a different underlying model to create the driving patterns.

## VI. SUMMARY

Taking the example of the German automatic toll system we have discussed the challenge to model the user interaction. Even with a simple model many parameters (e.g. probability distributions) need to be adjusted so as to achieve “good enough” simulation results. The use of a genetic algorithm simplifies the optimization i.e. adjusts the parameters as good as possible starting from the limited data available. The sheer number of parameters available poses a significant challenge even to a parallelized genetic algorithm. To us this suggests that the model of the user interaction is not yet expressed in the right way. In addition, “good enough” models depend on the context. In our example, a better model of the user interaction is needed to reproduce the intra-day behavior – e.g. to use the simulation to monitor everyday operations of the toll system.

Looking back at fig. 2 this article discussed the recently added model of a genetic algorithm. In future work the model of the user interaction should be split in two parts: A generic, domain-independent model of stochastic processes and its domain-specific application to generate driving patterns.

## REFERENCES

- [1] Barry W. Boehm. A view of 20<sup>th</sup> and 21<sup>st</sup> century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29. ACM, 2006. doi: 10.1145/1134285.1134288.
- [2] Robert L Glass. The standish report: does it really describe a software crisis? *Communications of the ACM*, 49(8):15–16, 2006. doi: 10.1145/1145287.1145301.
- [3] Michael Hobday, Andrew Davies, and Andrea Prencipe. Systems integration: a core capability of the modern corporation. *Industrial and corporate change*, 14(6):1109–1143, 2005. doi: 10.1093/icc/dth080.
- [4] Julia Numrich, Sascha Ruja, and Stefan Voß. Global navigation satellite system based tolling: state-of-the-art. *NETNOMICS: Economic Research and Electronic Networking*, 13(2):93–123, 2012. doi: 10.1007/s11066-013-9073-9.
- [5] Bernd Pfitzinger, Tommy Baumann, Dragan Macos, and Thomas Jestädt. Using simulations to study the efficiency of update control protocols. *47th Hawaii International Conference on System Sciences (HICSS)*, pages 5154–5161, 2014. doi: 10.1109/HICSS.2014.634.
- [6] ISO. ISO/IEC 19501:2005 unified modeling language specification, 2005.
- [7] Tommy Baumann. Simulation-driven design of distributed systems. *SAE Technical Paper*, (2011-01-0458), 2011. doi: 10.4271/2011-01-0458.
- [8] Bernd Pfitzinger, Thomas Jestädt, and Tommy Baumann. Simulating the German toll system: Choosing “good enough” driving patterns. In *Proceedings of the mobil.TUM 2013 – International conference on mobility and transport*, 2013. URL <http://www.mobil-tum2013.vt.bgu.tum.de/media/contributions/>.
- [9] Tommy Baumann, Bernd Pfitzinger, and Thomas Jestädt. Simulation driven design of the German toll system – profiling simulation performance. In *2013 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 923–926. IEEE, 2013. ISBN 978-1-4673-4471-5.
- [10] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976. doi: 10.1147/sj.153.0182.
- [11] Reinhard Fraenkel and Volker Hammer. Keine Mautdaten für Ermittlungsverfahren. *Datenschutz und Datensicherheit – DuD*, 30(8):497–500, 2006. doi: 10.1007/s11623-006-0259-2.
- [12] Dave Thomas and Andy Hunt. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999. ISBN 978-0201616224.
- [13] Tommy Baumann. *Automatisierung der frühen Entwurfsphasen verteilter Systeme*. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken, Germany, 2009. ISBN 978-3-8381-1266-4.
- [14] Barry Wilkinson and Michael Allen. *Parallel programming*. Prentice Hall New Jersey, 2<sup>nd</sup> edition, 2004. ISBN 978-0131405639.
- [15] Ibrahim H Osman and James P Kelly. *Meta-heuristics: theory and applications*. Springer, Berlin Heidelberg, 1996. ISBN 978-1-4613-1361-8.
- [16] Gabriel Alvarez. Can we make genetic algorithms work in high-dimensionality problems. In *Report 112*, Stanford Exploration Project, pages 195–212. November 2002.
- [17] Hartmut Pohlheim. *Evolutionäre Algorithmen – Verfahren, Operatoren und Hinweise für die Praxis*. VDI-Buch. Springer, Berlin, Heidelberg, 2000. ISBN 978-3-540-66413-0.
- [18] Ingrid Gerdes, Frank Klawonn, and Rudolf Kruse. *Evolutionäre Algorithmen: Genetische Algorithmen – Strategien und Optimierungsverfahren – Beispielanwendungen*. Springer-Vieweg, Wiesbaden, 2004. ISBN 978-3-322-86839-8.
- [19] Ronald W. Morrison and Kenneth A. De Jong. Measurement of population diversity. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutten, and Marc Schoenauer, editors, *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 31–41. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43544-0. doi: 10.1007/3-540-46033-0\_3.