Simulation Driven Development of the German Toll System – Simulation Performance at the Kernel and Application Level

Tommy Baumann, Bernd Pfitzinger, and Thomas Jestädt

Abstract. Simulation driven development – the idea of using simulation models as executable system specification in any phase of the system development process [4] – depends on the performance of the simulation model and execution framework. We study the performance issues of an existing large-scale simulation model of the German toll system using a discrete-event simulation (DES) model. The article first introduces the German toll system and the simulation framework developed to analyze the systems' behavior. To address the simulation performance the article describes a number of common performance limitations of several commercial and non-commercial DES simulation kernels. These performance limitations are addressed in kernel-level benchmarks. At the application-level a DES implementation of the German toll system is used to compare two commercial DES tools and several optimizations are introduced both on the simulation model and kernel level to achieve the necessary performance for a detailed and realistic simulation of a fleet of 750 000 trucks.

Tommy Baumann
Andato GmbH & Co. KG, Ehrenbergstraße 11, 98693 Ilmenau, Germany
e-mail: tommy.baumann@andato.com

Bernd Pfitzinger
Toll Collect GmbH, Linkstraße 4, 10785 Berlin, Germany
FOM Hochschule für Oekonomie & Management, Bismarckstraße 107,
10625 Berlin, Germany
e-mail: bernd.pfitzinger@toll-collect.de

Thomas Jestädt
Toll Collect GmbH, Linkstraße 4, 10785 Berlin, Germany
e-mail: thomas.jestaedt@toll-collect.de

M. Mach-Król and T. Pełech-Pilichowski (eds.), *Advances in Business ICT*,
Advances in Intelligent Systems and Computing 257,
DOI: 10.1007/978-3-319-03677-9_1, © Springer International Publishing Switzerland 2014

1 Introduction

Software evolution is a fact of life. Software-intensive systems become ever larger and to make matters worse include ever more distributed endpoints up to mobile and ubiquitous computing [11]. Introducing changes and new features to an existing system is both time consuming and error prone – one study [17] claims that the probability of critical problems due to poor design decisions is over 60% in the specification phase. Simulations are a vital step in the design of systems or the assessment of planned changes [3, 4] – reducing the inherent risk of ongoing system development and allowing for a faster system deployment. In addition simulations predict the dynamic system behavior which can become highly non-linear or chaotic even for simple systems [25].

To specify and evaluate the German toll system, a simulation-driven design approach has been selected [5]. The approach is characterized by applying modeling and simulation technologies in the early design stages, i.e. at at a time when most of the important design decisions have to be made. As a result both the systems and processes are specified in the form of executable models. The approach allows to validate and optimize the overall system architecture already in the specification phase – avoiding expensive integration issues in the subsequent implementation and integration phases.

Consequently, specification speed and quality is considerably increased while the system and product uncertainty is decreased. It is noteworthy that simulation-driven design not only refers to the system under design but also includes the surrounding design process, i.e. the process is also captured as an executable specification which allows automating design steps like architecture optimization, validation against operational scenarios and tracking of design decisions.

A prerequisite to applying executable models is a so called execution domain: In our context Discrete Event Simulation (DES, [19]) has gained significance. We choose DES as the execution domain of our simulation model (although in future work the behavior of the user interaction might better be modeled in an agent-based approach). DES is used in many industries, e.g. energy, telecommunications, production, logistics, avionics, automotive, business processes and system design. Inter alia DES is applied for dimensioning of resources, to answer questions about topology, scalability and performance regarding operational scenarios, to predict system behavior and to estimate risks.

Increasingly the performance in defining and executing models becomes vital due to the increased complexity of systems and processes as well as the customer requirement to create holistic, integrated, high accuracy models up to real world scale. Several use cases of simulations are only possible once the simulation performance is ‘good enough’: simulating the longterm dynamic behavior, iterative optimization loops, automatic test batteries, real-time models (higher reactivity to market demands and changes) and automated specification and modeling processes (including model transformation/generation) [28].

The outline of the article is as follows: Section 2 gives an overview of the automatic German toll system, the corresponding simulation model and typical simulation results. Section 3 introduces the performance properties of discrete event simulations and discusses appropriate performance metrics and benchmarking models. This is followed in section 4 by a discussion of the simulation performance and scaling of several DES tools for basic simulation operations. Using an existing microscopic holistic execution specification of the automatic German toll system [27] we describe several performance optimizations both on level of the simulation model and the simulation kernel in section 5. Section 6 provides a brief discussion of profiling a simulation run using internal or external profiler followed by the summary in section 7.

2 Executable Specification of the German Toll System

Toll Collect GmbH is the provider of the German electronic toll for heavy goods vehicles (HGVs). The system automatically collects the toll fees on federal motorways using an on-board-unit (OBU) installed in most of the trucks[1]. Currently there are more than 750 000 OBUs deployed, each determining the toll fees according to an up-to-date map of the chargeable roads using a GNSS receiver coupled to the vehicles' speed and directional data and communicating via GSM with the Toll Collect data center. In total, the HGVs drove $26.6 \cdot 10^9$ km on the chargeable federal motorways in 2012 [8] incurring a total of 4.36 bn€ [9]. For the application domain we use an existing simulation model of the German toll system [6, 28], a large-scale autonomous toll system [10].

Following the idea of simulation-driven design we use executable models to analyze and evaluate the behavior of the IT systems of Toll Collect GmbH. The simulation model of the Toll Collect system is used to predict the behavior of the current system as well as effects of changes to the system, especially to maintain the high level of accuracy (with an error rate of less than 1 in 1 000, [12, 36]) needed due to service-level-agreements. Changes to the Toll Collect system occur every day – in the past four years more than 15 major changes (releases) and more than 1 500 medium-sized changes were implemented.

2.1 Modeling of the Toll Collect System

The simulation model of the Toll Collect system consists of three blocks as shown in fig. 1 and an additional model for the user interaction (scenario generator). The model execution is controlled by a discrete event scheduler (in our example either MSArchitect [2] or MLDesigner [24]), responsible to initialize the vehicle fleet and to run the simulation. The vehicle fleet treats each OBU as an individuum with a distinct configuration and internal state. This state changes according to the simulation and the externally pre-calculated (statistically realistic) driving pattern of the

[1] An alternate mode of operations is available which offers the ability of manual booking.

OBUs [31, 32] containing their configuration (e.g. hardware and software versions), their power cycles and the toll charging instants.

Starting with the OBU and its driving pattern the model simulates the automatic communication between the OBU and the central systems either to transmit the tolls collected or to update the OBU state, geo and tariff data or software. Due to the arbitrary power cycles of the OBU and various resource restrictions (e.g. limited bandwidth, high latency, intentionally limited number of parallel connections for the central systems) it is common for data transmissions to be interrupted and subsequently recovered by application-level protocols.

The mobile data network includes provider specific transmission properties (e.g. bandwidth and latency) and resource constraints (e.g. actively managed number of simultaneous connections allowed). On the network layer the simulation includes the bandwidths and latencies observed for the various OBU hardware platforms and mobile network operators. The simulation includes the GPRS connection handling, the authentication handshake and IP address handling but does not include the IP network layer.

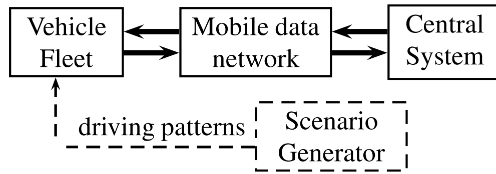

Fig. 1 High-Level simulation model of the Toll Collect system (upper half) and the model for the user interaction (scenario generator, lower half)

The block "Central System" includes the typical systems required to authenticate, receive and validate data transmission (e.g. firewalls, proxy servers, load balancers, database and application servers) each with their individual resource constraints. From a service management perspective the system is a sizeable service value chain spread across several service providers [26, 29, 30, 33].

To achieve realistic simulation results the model tries to include as many details as possible. Accordingly the vehicle fleet should include as many individual OBUs as in the real Toll Collect system (more than 750 000) with statistically realistic driving patterns for several consecutive months. In that way it is possible to simulate long-term behavior (e.g. a software update of the whole fleet) without resorting to scaling. The behavior of each OBU is implemented at a high-level of detail up to including the original source code of the OBU in the handling of internal state transitions.

2.2 Application and Results

The simulation model is used to determine the effects of the systems' configuration, e.g. on the progress of software updates or on the return to normal operations after outages of the central systems. This can be extended to determine the optimal system

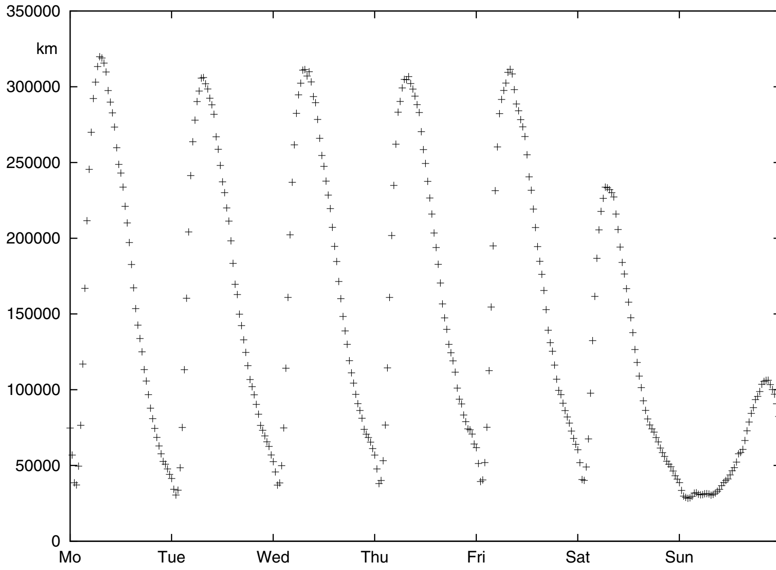

Fig. 2 Weekly driving pattern of a vehicle fleet of 140 000 HGVs. Each point represents the chargeable kilometers driven within a 30 minute period.

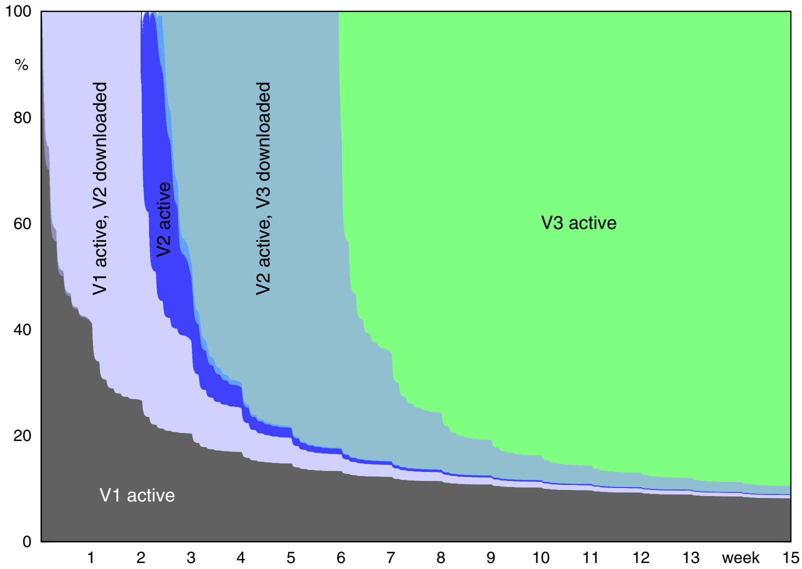

Fig. 3 Two consecutive map data updates of a fleet of 140 000 OBUs over a 15 week period, the update is downloaded and activated only after the start of the validity period

configuration e.g. considering the trade-off between operational costs and the cost of financing (of not yet processed toll fees) [27, 28].

The execution of the Toll Collect model using MLDesigner takes about 6 hours of single-core CPU time (Intel Xeon X5670 at 2.93 GHz) to simulate 16 weeks with a fleet size of 140 000 HGVs (corresponding to a 1:5 scaling). The pre-calculated driving pattern changes according to the day of week (see fig. 2) and is based on a statistical analysis of the driving pattern over a 15 week period (in early 2011). The weekend and Sunday truck ban on German highways is clearly visible in fig. 2.

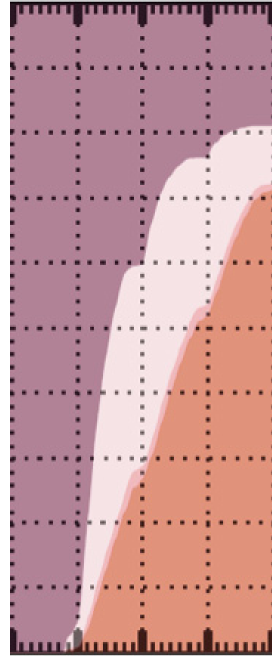

Fig. 4 Simulated software update from the initial version (purple) to the new version (orange) [31]

Additional data from the Toll Collect test fleet ($> 2\,000$ HGVs) is used to parameterize the number and duration of power cycles. The example uses an average of 1112 power cycles per OBU and year (with a minimum duration of one minute per power cycle).

This microscopic simulation model is also used to determine macroscopic effects, e.g. the periodic update of map and tariff data on the OBU. The update process is initiated by the OBU which periodically checks for the availability of updates and schedules the download of new updates randomly prior to the start of the validity period of the new data. Fig. 3 shows the result of two consecutive updates of the map data, where each OBU has one version of the data installed and possibly either knows about the existence of a new version or has it already downloaded (but not yet activated). With the start of the validity period of the new version OBUs that had it previously downloaded will immediately switch to the new version (provided the OBU is powered on). OBUs that are unable to download the new data in time (e.g. OBUs staying outside of the German mobile data network coverage) will try to retrieve the update as soon as the power restored to the OBU and the OBU is within reach of the German mobile data network. Across the whole vehicle fleet we observe that about 10% of all OBUs do not connect to the data center within a given 15 week period.

Since [6] we have switched to use the MSArchitect simulation framework to achieve simulation runs at a 1:1 scale: From the process perspective the simulation model covers business and system processes differing at least 7 orders of magnitude in time: All major technical processes with durations of one second and longer are included in the model aiming to predict the dynamic system behavior of fleet-wide

updates (taking weeks to months, fig. 4). In fact, the model includes some processes with higher temporal resolution (down to 50 ms for the connection handling in the DMZ) and is used to simulate all updates occurring over a whole year. Using the Pearson correlation as metric to compare the simulation results with the observed update rates between April 2012 and January 2013 we find the correlation to be above (better than) 0,994 (see tab. 1). The current investigation is to validate the simulation model using additional metrics and a time-scale of one hour [32] (instead of one day).

Even on the application level the user interaction (scenario generator) creates a large number of events to be processed by the simulation logic. On average each OBU will be powered-on for 16% of the time and process tolls for 32 000 km annually ([7], one toll event per 4.2 km on average [12]) spread across some 1 300 power cycles (including three times as many periods of mobile data network). Of course, many more events are created from within the application logic, e.g. to forward tolls to the central systems or to run error recovery protocols in the case of network unavailability.

Table 1 Comparing fleet-wide updates (simulation results vs. data from Apr 2012 to Jan 2013)

	correlation
software	0.99963
geo data	0.99572
tariff data	0.99475

2.3 Simulation Performance

To achieve realistic simulation results we decided against the use of a simplified simulation model (as compared to the real-world system) and aim for a 1:1 scale, i.e. more than 750 000 individual OBUs within the simulation and a realistic behavior on the network layer. Therefore the typical time-scales within the simulation are on the order of 100 ms. However, the business processes of interest have a typical time-scale of one to two months: e.g. map and software updates are intentionally spread over many weeks to be able to reach HGVs that are operating outside of the German mobile network coverage.

As a consequence the simulation performance must allow to simulate at least three consecutive months of a realistic driving pattern with a full-size vehicle fleet. Using the simulation as part of the design process or to validate changes to the systems' configuration necessitates that a typical simulation run delivers results within the business day. Unfortunately, the tools used do not yet allow the automatic distribution of the simulation across several CPUs (or even CPU cores).

The initial implementation of the simulation model with MLDesigner led to various performance bottlenecks due to the large number of OBU objects and scheduled events within the simulation. The extraction of the OBU logic from the simulation model to conventional C++ classes alleviated the performance degradation and the memory usage. Changing the model implementation and switching to the MSArchitect simulation framework we were able to increase the fleet-size to realistic scales. A prerequisite is a detailed understanding of the performance issues present

in the simulation model and the tools used for execution. Therefore the remainder of the article focuses on benchmarking of simulation tools or identifying performance hotspots in a given simulation model.

3 Evaluation of Discrete Event Simulation Performance

3.1 Importance of Performance

It is well known [20, 21] that software-intensive systems evolve towards ever increasing complexity – fulfilling more user requirements, interfacing with additional other systems and of course requiring ever more lines of code. Modeling and simulation methodologies and technologies [5] can be applied to design, analyze, evaluate, validate and optimize such systems – far in advance of the actual implementation. Executable models are created as blueprints of the new system and are used as functional ("virtual") prototype. At an early stage of the design process these virtual prototypes give insight into the systems' behavior e.g. regarding the scaling properties, the advantages and disadvantages of the system topology. At any time simulations can be used to explore operational scenarios (especially those exceeding the systems' specification) and the inherent risks (operational and procedural).

In this context simulation performance needs to keep up with the enormous complexity increase of executable models, which in turn follows the complexity increase of systems and processes. In addition, executable models should include a high level of detail. Together with systems including a large number of active components (e.g. users, machine-to-machine networks) this results in a complex simulation model – both from a static and dynamic perspective.

In a business context, the simulation is often part of an optimization process, i.e. the optimal solution is determined by iterative optimization loops. In that case many steps consisting of a complete simulation run (possibly including test batteries) need to be evaluated to determine the optimal solution. Of course this approach is beneficial only if the simulation results are both reliable and available well in advance of traditional software engineering approaches. Hence simulation performance in terms of speed and memory consumption and its benchmarking became a critical aspect in system design.

3.2 Performance Benchmarking

There exist several approaches for benchmarking simulation performance, especially kernel benchmarks and application benchmarks are common [34, 35]. A kernel benchmark consists of several, typically smaller test cases where each test case stresses a single elementary function of the simulation kernel (see fig. 5). Therefore kernel benchmarks are useful to analyze the built-in performance of low-level mechanisms. The results are typically weighted according to their importance for a given application domain – however, the predictive power of kernel benchmarks for real-world application performance is limited. To compare application performance,

the benchmark measurements include a number of real-world examples from the application domain. The selected applications should exhibit different characteristics and represent typical challenging workloads.

In the Toll Collect example we started with an existing simulation model using a given simulation tool (MLDesigner). To achieve the necessary performance (as outlined in section 2.3) both kinds of benchmarks were used: A low-level analysis of the simulation kernel allows to identify performance bottle-necks in the existing simulation model and tool. In addition the kernel benchmark is easily adopted toward different simulation tools. Section 4.2 gives a description of the kernel benchmarks used to benchmark a total of five different DES simulation tools, followed by a comparison and discussion of the kernel benchmark test results.

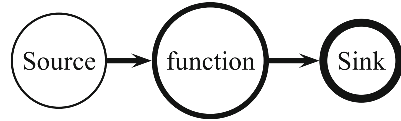

Fig. 5 Basic test model for kernel-level benchmarks used to test elementary functions

As a consequence of the kernel benchmark results the Toll Collect simulation model was ported to a second simulation tool – requiring considerable effort and expertise (both of the simulation model and tool). Having the same simulation model implemented for two different simulation tools allows for direct comparison and benchmark at the application-level (as shown in section 5.1).

4 Kernel-Level Benchmarks

DES simulations are typically split into the simulation environment and the simulation model. The simulation environment itself is used to create models (using an interactive and graphical user interface), to execute existing simulation models and possibly also to visualize the progress and results of a simulation run. The simulation model itself contains all static model entities, their relationships and methods to handle events during the model execution. Thus the simulation model determines the dynamic properties of a simulation run, e.g. the number of entities present during the model execution and the number of events created.

4.1 Simulation Kernel Benchmark Tests

Since the execution control resides with the simulation kernel, the implementation of event handling (especially the future event list (FEL) and its update mechanism), the data and memory handling (e.g. pass-by-reference vs. pass-by-value, garbage collection) and the use of caches determines the simulation kernel performance. Similar to [13] we include the following elementary factors in our kernel-level benchmarks:

- FEL management: The event scheduling mechanism is the core of any discrete event simulation determining the dynamic behavior of the simulation. At any given time the model entities create new events scheduled to take place in the future and sometimes cancel existing future events as well. The crucial performance factor of a DES simulation kernel is therefore the handling of the future event list. Its management can be more time consuming than the actual data manipulation.
- Memory and data type management: The allocation and maintenance of tokens and memory for dynamic model entities is an important issue. The event handling will inevitably deal with the creation and deletion of a large number of events, events passed between model entities usually need to transport additional (application-level) data between the entities, possibly necessitating the casting between data types (incurring an additional overhead). The efficient storage of the information will directly affect the simulation performance. A *pass-by-value* approach will incur additional overhead (due to the necessity of duplicating the data). A *pass-by-reference* implementation of the FEL management algorithms processing the tokens representing an events should yield better performance – especially if the simulation entities are only referenced from the event tokens. Dealing with memory allocations can be improved by the use of caching mechanisms.
- Pseudo-random number generator performance: A basic requirement of DES simulation execution is the ability to use random numbers to achieve a "non-deterministic" behavior. A typical DES tool includes generators for several different random number distributions. It is critical to be able to use large streams of pseudo random numbers.
- Arithmetic operations: The actual data manipulation is given by arithmetic operations either in an imperative or functional language. This programming language needs to be executed at runtime and can become a performance bottleneck if the chosen programming language does not allow compilation to the underlying CPU architecture.

In addition the ability to generate reports or to export reporting data is a basic requirement for any DES simulation. Creating the reports and the underlying data can incur considerable additional computational expense. However, the reporting requirements are typically driven by the application domain. Therefore we do not include reporting in our kernel-level benchmark.

4.2 Simulation Kernel Performance Tests

We present five different test models for DES simulation kernel benchmarks, addressing all elementary factors presented in section 4.1. These models are applied later to investigate and compare DES kernel performance. The models have been kept simple in order to assure universality regarding different kernels/tools and to avoid possible side effects.

4.2.1 Simulation Scaling

A simulation model with several hierarchy levels (fig. 6) is simulated in a sequence increasing the total number of events, while the size of the future event list remains fixed. This test determines whether the FEL performance is affected by the FEL size. The test used a clock interval of one and the number of events processed increased from $0.1 \cdot 10^6$ to $10\,000 \cdot 10^6$ events.

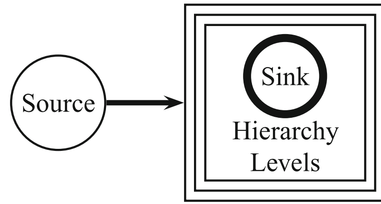

Fig. 6 Test model for simulation scaling

4.2.2 FEL Size Scaling

The second test uses the generic simulation model of fig. 5 with a delay-function. The delay is used to easily configure the (average) number of events waiting in the future events list with minimal variance, while the total number of events processed remains constant. This test examines the overall performance of the FEL algorithm and data management. We used a uniform distribution of events in the FEL list. Of course, the test can be extended toward non-uniform events distributions, in order to check adaptability of the FEL algorithm on different events densities.

For this test we use a clock interval of one, a fixed number of processed events ($300 \cdot 10^6$) and configure the delay-function to produce a given size of the future events list (between 10^6 to 10^7 events, constant over single experiment).

4.2.3 FEL Adaption

This test extends the future events list size during one test run by changing the parameter of the delay-function dynamically during the simulation execution. The test extends the previous test model by additional single events used to change the parameter of the delay-function (see fig. 7). As a consequence the size of the FEL changes during the test run forcing the simulation kernel to adapt the FEL size (e.g. allocating and deallocating memory) during the simulation run.

The test uses a clock interval of one and a dynamic delay-function parameterized to give a dynamic FEL size of $1\,000 - 10^6 - 100 - 10^7 - 10$ events during the simulation run. In total one test run consists of $200 \cdot 10^6$ processed events.

4.2.4 Memory and Data Type Management

The test creates large data arrays of different sizes and passes the data through the simulation model in sequential or parallel order as depicted in figure 8. When executing the model the memory management of the simulation kernel should recognize the passing of unmodified data and use references to this data. Ideally only one datum should be created and send as reference through the model. As

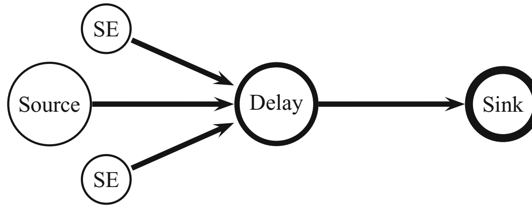

Fig. 7 Test model for FEL adaption with additional single events (SE)

long as the delays are set to zero, no difference between serial and parallel passing should be recognizable. The test uses a fixed number of nodes (delay blocks) either in a parallel or serial configuration and data arrays with $1, 0.5 \ldots 2 \cdot 10^6$ entries.

4.2.5 Random-Number Generator Performance

A large number of random values is generated using different distributions. The model uses a constant function as a reference to measure relative performance of the built-in pseudo-random-number generators. The test computes $20 \cdot 10^6$ random numbers of different random number distributions (normal distribution, Poisson distribution and exponential distribution).

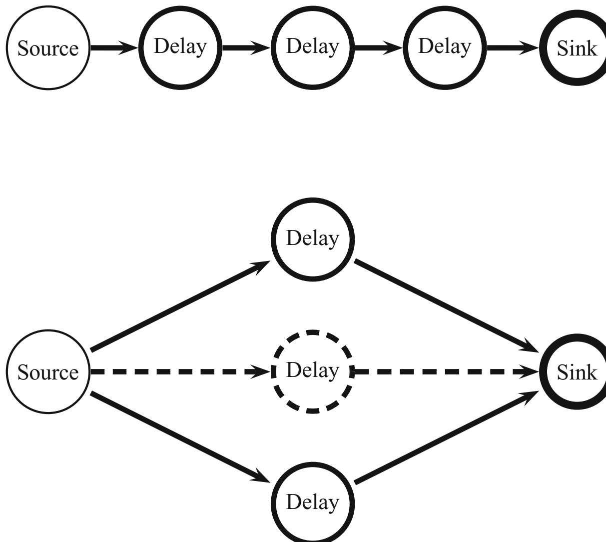

Fig. 8 Test model for memory and data type management for sequential (top) and parallel (bottom) processing

4.3 Evaluation of Simulation Kernel

Each test model is simulated with a set of simulation parameters using different system design tools. Currently more than 80 tools listed for DES [1]. We selected six system design tools for evaluation: Ptolemy II, Omnet++, AnyLogic, MLDesigner, SimEvents and MSArchitect. All tools were run in serial mode (DES, not PDES) on an Intel Core i7 X990 at 3.47 GHz with 24 GiByte RAM using either Windows 7 Enterprise (64 bit) or openSuse 11.4 (32 bit, kernel 2.6.37.6). Performance data was recorded with Perfmon on Windows and sysstat and the Gnome System Monitor on the Linux system.

Fig. 9 gives the results of the Runtime Scaling test. The upper chart shows the event processing performance for different simulation lengths and the bottom chart the private memory consumed during simulation. The tests show that neither the

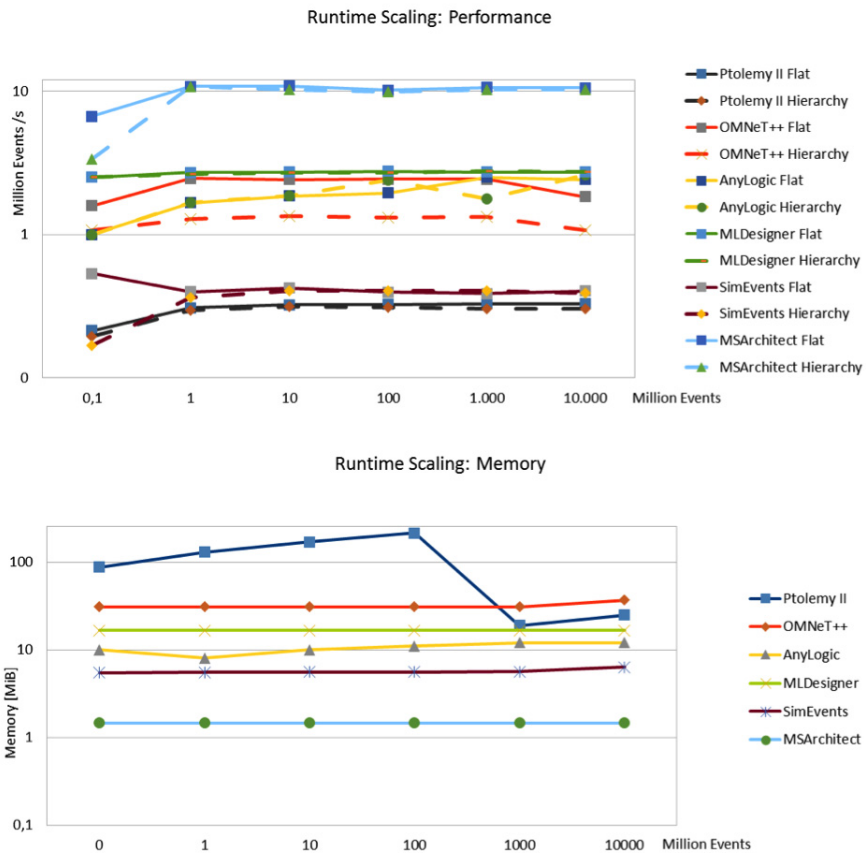

Fig. 9 Runtime performance (top) and memory usage (bottom) scaling of different DES tools

memory consumption nor the event processing performance is affected by increasing the simulation runtime. Looking at the sensitivity of running the tests with additional hierarchy levels we find that only OMNeT++ is sensitive to the additional hierarchy levels. However, from the test results it is already obvious that the different tools vary in event processing performance by an order of magnitude: MSArchitect provides the highest speed. MLDesigner, AnyLogic and OMNeT++ provide 25% of the speed (compared to [6] the MSArchitect performance improved by more than 30%). Ptolemy II is twenty times slower. Looking at the memory usage during the simulation the difference between the tools is again more than an order of magnitude – the slowest tool using the most memory and the fastest tool using the least. Two of the tools (Ptolemy II and AnyLogic) are based on the Java programming language, where explicit memory deallocation is not possible. Apparently the Ptolemy II test run triggers the JVM garbage collection during the simulation run and is able to free 90% of its memory. As a result Ptolemy II memory consumption is then comparable to the next three simulation tools. The AnyLogic test run starts already with a much lower memory consumption than Ptolemy II and no effect of JVM garbage collection is visible.

Fig. 10 gives the results of the FEL Size Scaling test. As expected, a systematic performance decrease can be observed with increasing FEL size, due to the increasing overhead for FEL management. Most of the tools tested initially start with relative constant performance (on a log-log scale). With increasing FEL size three of the five tools develop drastic performance degradation. This coincides with a rapid grow of memory consumption with increasing FEL size. We propose that the performance reduction is correlated with increased FEL memory usage due to a performance penalty of calendar queue based schedulers for large queue sizes. Again, Ptolemy II has the lowest performance in this test. OMNeT++ is nearly not affected in the considered FEL size interval. In absolute numbers, MSArchitect has the best test performance and the lowest memory usage until FEL size 10^6. Subsequently the memory usage of MLDesigner is lower since MSArchitect runs in 64 bit mode which in fact means a higher memory demand due to larger address ranges. But in our benchmark MLDesigner stops working for FEL sizes above $15 \cdot 10^7$. We tested MSArchitect successfully with a FEL size of 10^8.

In the FEL Adaption test the simulation kernel is subjected to a varying demand to its FEL. Beyond the runtime needed for the test the main result is the memory consumption during the test run as given in fig. 11. The simulation took 2 276s with MLDesigner, 673s with AnyLogic, 192s with MSArchitect, 395s with OMNeT++ and over 2 hours with Ptolemy II. During that time the dynamically changing memory usage varies widely between the different tools. Most tools tend to allocate memory in chunks visible as steps in fig. 11. Again, Ptolemy II is the slowest tool in comparison and also requires more memory than any other tool in the benchmark. The memory usage of OMNeT++ indicates the ability to dynamically free already allocated memory. However, this simulation tool also allocates considerably more memory than any of the other tools for a brief period of time during the test run.

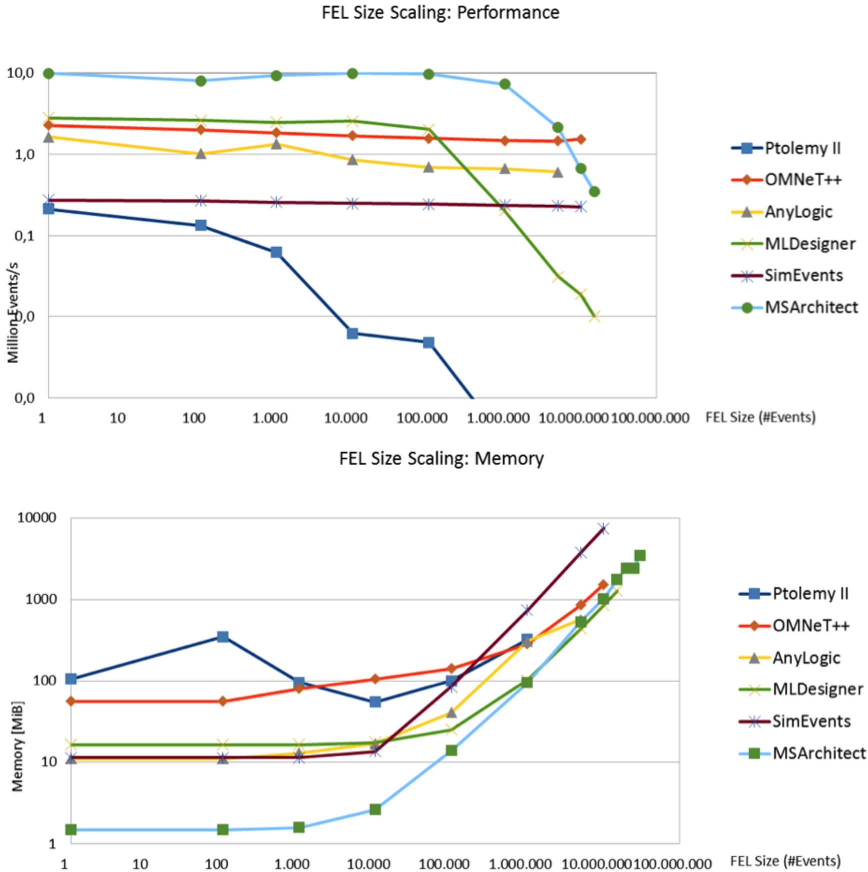

Fig. 10 Future event list size scaling test results for runtime performance (top) and memory usage (bottom) scaling of different DES tools

The Data Type Management test passes large arrays of data through the simulation running either in a parallel or serial configuration. The memory consumption during the test run is shown in fig. 12. Most tools handle serial and parallel passing of token data in a different way, which can be recognized by the gap in memory consumption between both serial and parallel versions. Ptolemy II and MSArchitect do not show a difference between the parallel and serial version, only references are passes when only delays are used. However, Ptolemy II requires more memory and shows a different behavior according to the memory allocation: a large portion of the memory is allocated at initialization time with standard modeling elements.

The last test is the Random Number Generation test. As depicted in fig. 13 the performance does not depend on the type of the generated distribution, since there are only minor differences to the generation of constant numbers. Again, Ptolemy II is an order of magnitude slower than OMNeT++, AnyLogic and MLDesigner in this

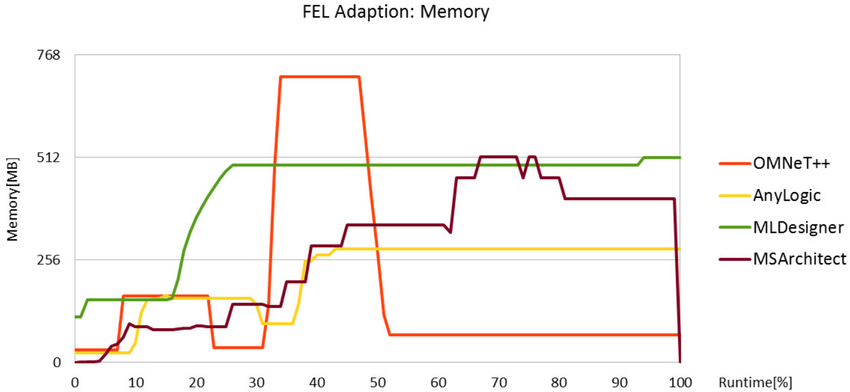

Fig. 11 Test results for future events list adaption

Fig. 12 Test results for the memory consumption during Data Type Management test case

test. MSArchitect gives the highest performance compared to the other tools. It is not clear whether the pseudo-random number generator (PRNG) algorithm differs between the five simulation tools or if the PRNG performance is adversely affected by event management overhead. Since this test relies on the correct implementation of the PRNG, i. e. we do not check the statistical quality of the random numbers generated, the test results might not be fair if one of the tools were to use low-quality but high-speed generators.

It can be concluded, that Ptolemy II is inferior in all simulation kernel benchmarks performed. MLDesigner is equal to or better than AnyLogic in all categories but FEL adoption. Due to the utilization of the JVM, AnyLogic requires more memory in equivalent models and therefore scales worse with increasing FEL size. Both,

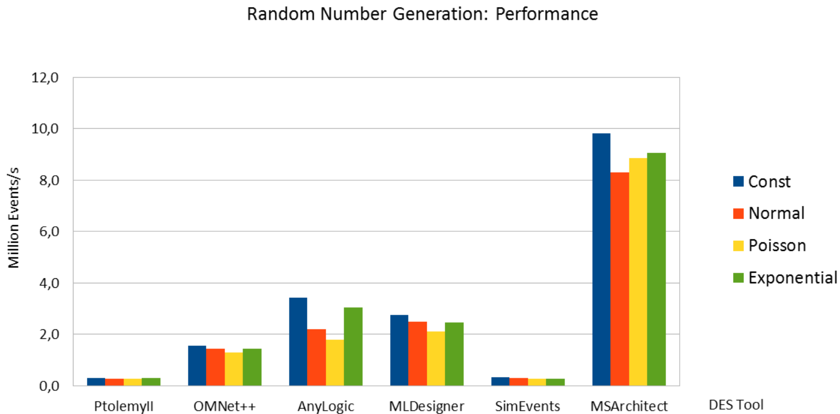

Fig. 13 Test results for random number generator scaling

MSArchitect as well as OMNeT++ show the best performance in some categories. MSArchitect is the fastest simulation kernel in most categories and requires least memory for data handling.

5 Enhancing Simulation Performance of the Toll System Model

While section 4 focused on the performance evaluation of DES kernels we now focus on the application-level performance, i.e. how to specify *efficient* executable DES models.

5.1 Evaluation of Model Architecture

A simulation model can be thought of as a (simplified) copy of an existing or imaginary system, created for a certain purpose. The model and the process of creating the model are a key to learning and communicating about the system itself. This implies that the right level of abstraction needs to be found so as to include only the system behavior relevant to the models' purpose. Bearing this in mind the most important rule in designing efficient models can be derived: *The level of detail always follows the model purpose*. For instance it makes a huge difference for choosing the appropriate level of detail when designing a data transmission model compared to modeling a rather abstract business process. Of course, any useful model must be connected in some way to the reality. A second point directly connected to that rule is to focus on measurable system behavior. Otherwise the model could become worthless when using it for analyzes and optimization.

Designing the simulation model directly affects the runtime properties (e.g. performance and memory consumption). From a technical point of view the performance of a model can be improved considerably by addressing several issues:

- the number of simulation entities present during the simulation run,
- the data transport between model components to reduce the number of DES events and
- the execution time and memory consumption of the model components.

Hence whenever possible, highly interacting model components should be merged together to avoid the time consuming data exchange via the simulation kernel. In addition data should always be transmitted in form of references/pointers (*pass-by-reference*). References are values that enable indirect access to a particular datum, indistinct from the data itself. They are used to efficiently pass large or mutable data. In that way the time-consuming and unnecessary copying of data is avoided.

The simulation kernel benchmarks in the previous section identified the future events list as a key factor in the kernel performance. The tests were designed to continuously create new events leading to different FEL sizes. Creating, scheduling and passing events is certainly the key feature of DES simulation kernels. However, a simulation model sometimes needs to be able to cancel scheduled future events before they are executed. Many simulation tools lack a good implementation for canceling events from the FEL, possibly needing to traverse the whole FEL in the search of the canceled event and possibly triggering memory reorder after removing the canceled event from the FEL. Obviously, simulation runs with a large FEL are more affected.

The performance of simulation models can be improved by transferring part of the simulation model to existing standardized model components or even extending the simulation tools' existing catalog of standard components.

5.2 Performance Enhancement to Our Solution

As the project of modeling the German toll system was launched our team had no clear picture of the coming performance issues: Existing simulation models of HGV tolling systems both at Toll Collect and in the literature were limited to a few thousand simulated HGVs [16, 22] and reaching 500 000 HGVs over a 4 week period [23]. Our model aimed to include a more detailed behavior and a vehicle fleet almost two orders of magnitude larger (comparable but still larger than simulations of metropolitan car traffic, e.g. [14] using 200 000 drivers with a shorter simulated time frame).

After putting together and validating the basic DES model in MLDesigner, including the dynamic behavior of the vehicle fleet, mobile providers and central system we tried to scale up to the real world situation. This meant to run simulations of vehicle sizes of up to 750 000 HGVs over a simulated time period of at least 3 months. The disappointing simulation performance results are shown in the second column of table 2. The desired scenario took about 49 million seconds, over 6 times *slower than reality*, an unacceptable result.

By transferring the model from the system design tool MLDesigner to MSArchitect the simulation performance could be increased dramatically. On the one hand the throughput of events is about 3.5 to 4 times higher in MSArchitect (as confirmed by the kernel benchmarks in section 4). On the other hand we recognized huge performance issues in the management of complex data structures in MLDesigner. We analyzed the differences by comparing the simulation performance of using MLDesigner data structures versus using pointers to external C++ classes for data transport (the default in MSArchitect). In total the transfer of our simulation model from MLDesigner to MSArchitect brought a 120-fold speed increase.

Table 2 Simulation performance for the Toll Collect example with a simulated time period of three months

	Runtime [s]	
fleet size	MLDesigner	MSArchitect
70 000	0.25 M	900
700 000	49.00 M	8 700

On top we redesigned our model architecture. First of all removing all “cancel event” operations from the model – being rather expensive operations in both simulation tools. The canceling of events was replaced by introducing an additional boolean tag to store whether the next receiving event is ignored or not. By doing so the overall amount of events in the FEL is increased and more memory is needed but time consuming cancel operations can be avoided.

Next we removed several retry processes between vehicle fleet and mobile data network providers and merged heavily interacting model components to minimize data transport across the simulation kernel. In addition we switched to the data structure mechanism of MSArchitect which automatically uses references when sending or receiving unchanged data tokens.

In total a further significant performance increase could be achieved. The rightmost column of table 2 shows the results of two different scenarios executed with MSArchitect. Simulating the scenario stated above (750 000 HGVs over a three months period) took about 8 700 seconds. Thus the simulation speed could be increased by a factor of 5 630 compared to the initial runs using MLDesigner. It is noteworthy that the model used with MSArchitect also includes additional additional functionality of the German toll system.

6 Profiling of the Simulation Model

To evaluate the application-level simulation performance of our model of the German toll system, we use both the kernel logging capabilities of MSArchitect and an external profiling application (Intel VTune). Kernel logging allows to count the number of calls of atomic models as well as the total number of samples (corresponding to a processor cycle). The external profiler allows measuring the space complexity (memory), the time complexity (duration, CPU time) and the usage of particular instructions of a target program by collecting information on their execution. The most common use of a profiler is to help the user evaluate alternative

implementations for program optimization. Based on their data granularity, on how profilers collect information, they are classified into event based or statistical profilers [15]. We've selected the statistical profiler Intel VTune Amplifier XE and connected it to the generated C++ runtime representation of our model. As test environment, an Intel Core i7 K875 at 2.93 GHz with 8 GiByte RAM and Windows 7 Professional (64 bit) installed has been used. To profile the simulation model we take the simulation scenario used to verify the simulation model against real-world data (Apr 2012 to Jan 2013).

6.1 Profiling with MSArchitect

In a first step we apply the kernel logging capabilities of MSArchitect resulting in a file with profiling information at the end of the simulation run. Tab. 3 shows an excerpt of the file, containing all atomic blocks relevant to analysis (15 out of 65). Since during simulation all composite blocks are resolved to directly communicating atomic blocks, the table only contains atomic blocks of the simulation model. For each atomic block the table shows the number of calls, the accumulated count of samples, the time required in relation to other atomic blocks and the samples needed for one call.

First of all, the atomic block `AccessSessionStateSwitch` is striking, since it consumes a large amount of time due to the high number of calls. The block is responsible for switching OBU data structures in response to its state to one of the output ports. As the block switches between 34 states, 539 samples per call are acceptable. Nevertheless the number of calls could be reduced for performance

Table 3 MSArchitect kernel performance logging results

Atomic Block	Calls [M]	Samples [G]	Time [%]	Samples per Call
`AccessSessionStateSwitch`	19 980	10 760	10,89	539
`ExternDStxt`	0,0007	9 565	9,68	13 665 M
`StaHandling`	482	6 503	6,58	13 483
`EinzelbuchungsHandling`	4 660	6 442	6,52	1 382
`IpAutomat`	7 323	5 749	5,82	785
`Delay (Standard)`	8 874	5 522	5,59	622
`CheckComponentState`	7 363	3 859	3,91	524
`NetzverlustHandling`	3 020	3 479	3,52	1 152
`AccessSessionStateWrite`	5 841	3 215	3,26	551
`MfbSwitch`	5 525	3 196	3,24	579
`Nutzdaten`	3 563	2 694	2,73	756
`TcmessageCopy`	2 030	2 010	2,03	990
`TcpAutomat`	1 291	1 533	1,55	1 187
`TimedAllocate`	2 570	1 484	1,50	578
`SimOutObuVersions`	0,017	1 509	1,53	89 M

improvement by changing the model architecture – especially once the model is ported to the parallel DES core, it is an obvious block for introducing parallelism.

The next conspicuous atomic block is `ExternDStxt`, reading the pre-generated files provided by the scenario generator model as ASCII file. The block consumes 13 665 M samples/call and is rarely executed (700 times, i.e. twice per simulated day) resulting in a time consumption of 9,681% of the time. In order to reduce the load, scenarios should be computed on the fly. The atomic block `StaHandling` is responsible for generating and controlling status requests, which may result in update processes. The block consumes 6,581% of simulation time. We see potential for improvements in changing the implementation (e.g. conversion of formulas to save operations, replacing divisions by multiplications with reciprocal and using of compare functions from standard libraries).

With 4 660 M calls `EinzelbuchungsHandling` is a frequently executed atomic block. After analyzing the implementation we find 1 382 samples/call acceptable. The block depends on the random number generator and would benefit from faster random number generation algorithms. The atomic block `SimOutObu Versions` cyclically writes the software, region and tariff version of all OBUs to an output file. In our scenario we simulate 50 weeks and write data every 30 minutes, resulting in 16801 calls. 89 M samples/call seems to be quite costly and offers room for improvement.

In summary the simulation of the scenario took 98 811 263 M calls. Of these, the model components consumed 84,51% and the simulation kernel (logical processor) 15,49%.

6.2 Profiling with Intel VTune

In the second step we apply the profiling application Intel VTune [18]. The external profiler catches the activities of both the simulation kernel and the simulation model (denoted as “K” or “M” in tab. 4).

An excerpt of the results is shown in tab. 4. For each function the CPU time in percent, the amount of needed instructions (instructions retired), the estimated instruction call count, the instructions per call on average and the last level cache miss rate (0,01 means one out of one hundred accesses takes place in memory) is shown.

Most of the CPU time is consumed by kernel functions responsible for data transport. These functions are grouped by component (resp. namespace `msa.sim.core`, denoted as “K” in the first column of tab. 4). In total these functions consume 61,1% of the CPU time. Conspicuous is the relative high last level cache miss rate of function `EventManager.enqueueEvent` with 3,2% and the number of instructions needed per call `LogicalProcessor.mainLoopFast` with 2 379. However, the number of calls depends on the dispatch of data within atomic model components, which are grouped in form of user libraries. In our model we have two user libraries: GPRSSimulation (`GPRSSimulation.Components.Atomics`, denoted as “M” in the first column of tab. 4) and Standard (`msa.Standard.`

Table 4 VTune profiling results for simulation kernel (K) and model (M) ordered by CPU time. Shown are the CPU instructions retired (IR), estimated call count (eCC), instructions per call (IPC) and last level cache miss rate (MR).

	Function	Time [%]	IR [G]	eCC [M]	IPC	MR [%]
K	Port.send	9,0	44	689	65	0,4
K	EventManager.enqueueEvent	7,7	21	92	237	3,2
K	LogicalProcessor.mainLoopFast	7,0	17	7	2 379	0,3
K	EventManager.dequeueEvent	6,3	104	2 517	41	1,1
K	big._mul<unsigned int>	5,0	103	2 611	40	0,3
M	StaHandling.Dice	4,8	12	11	1 097	0,1
K	EventManager.scheduleEvent	3,5	53	1 286	42	0,2
K	Any.extractToken	3,0	70	1 805	39	1,7
K	Pin.popFrontToken	2,8	49	1 234	40	0,2
K	EventManager.bucketOf	2,7	17	327	55	0,0
K	Any.operator=	2,5	54	1 403	39	0,2
K	Any.create	2,3	64	1 689	38	0,4
K	random.tr1.UniformRng.getNextV	2,2	39	961	41	0,2
K	Any.doClear	2,1	22	497	45	0,2
K	Tokenizer.nextToken	1,8	22	606	36	0,3
K	TemplatePort<Tcmessage>.receiveToken	1,7	29	726	40	0,3
K	TemplateTypeInfo<EventData>.createToken	1,6	84	2 326	36	2,1
M	AccessSessionStateSwitch.run	1,5	13	287	48	0,2
M	EinzelbuchungsHandling.run	1,4	5	66	87	7,2
M	IpAutomat.run	1,4	7	103	70	3,4
K	Pin.popFront	1,3	6	89	74	0,3

`Control`). The latter is a support library included in MSArchitect. Combined they are responsible for 20,1% of CPU time consumption. Performance critical and starting point for improvement is the function `StaHandling.Dice` with 1 097 instructions per call and a CPU time consumption of 4,80%.

Both, kernel logging and profiling showed that most of the resources are utilized by functions responsible for data input/output (data mining) and functions responsible for transmission and processing of tolling information. By doing the analysis we located multiple components with potential for optimization, e.g. `AccessSessionStateSwitch` and `StaHandling`. Furthermore we came to the conclusion to generate scenarios on the fly since the reading of pre-generated scenario files is as time consuming. Relating the resource utilization of model components to real-word applications we could recognize a weak correlation. Model components like `STAHandling`, `EinzelbuchungsHandling` and `IPAutomat` are abstractions of important real word system applications and crucial to performance in both worlds.

7 Summary and Outlook

Extending [6] we have shown how to analyze the performance of DES simulations: Generic benchmark test-cases allow a simple and direct comparison of different simulation tools. Not surprisingly the tools differ vastly as to their time and memory consumption. However, the benchmark results cannot be transferred to the application domain: The workload generated by a given simulation model determines in large part its performance. Taking an existing simulation model of a large-scale technical system we performed an in-depth performance analysis for one simulation tool using both the performance analysis methods provided by the simulation kernel and an external profiler with access to the CPU hardware profiling support.

Both profilers immediately identify the same bottleneck: Reading the ASCII-formatted pre-calculated driving patterns from disk. Further analysis showed that calculating the driving patterns is less time-consuming than storing them on disk. Consequently the simulation model is now integrated with the scenario generator. This in turn will allow implementing an optimization algorithm to fit the driving patterns to the observed system behavior – a feature that we expect to drastically improve the accuracy of the simulation results for the short-term behavior [32].

The hardware profiler catches both the application-level methods as well as the atomics provided by the simulation kernel (with or without access to its source code). Taking the workload generated by this application we can start to tune the behavior of the atomics to improve the overall performance. Looking e.g. at the cache miss rate we find some simulation kernel routines and several application-level methods with a considerable probability of needing access to the main memory. We take this as starting point for future improvements.

MSArchitect, the simulation kernel used in the application benchmark, is currently extended to allow the automatic model reduction and (semi-) automatic parallelization of simulation runs. The single-core benchmark performed here will be the baseline to measure the improvements against.

References

[1] Albrecht, M.C.: Introduction to discrete event simulation (2010), `http://www.albrechts.com/mike/DES/Introduction` (accessed April 10, 2012)

[2] Andato GmbH & Co. KG: MSArchitect, `http://www.andato.com/` (accessed May 12, 2013)

[3] Banks, J., Nelson, B.: Discrete-Event System Simulation. Prentice Hall (2010)

[4] Baumann, T.: Automatisierung der frühen Entwurfsphasen verteilter Systeme. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken (2009)

[5] Baumann, T.: Simulation-driven design of distributed systems. SAE Technical Paper (2011-01-0458) (2011), doi:10.4271/2011-01-0458

[6] Baumann, T., Pfitzinger, B., Jestädt, T.: Simulation driven design of the German toll system – evaluation and enhancement of simulation performance. In: 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 901–909. IEEE (2012)

[7] Bundesamt für Güterverkehr: Maut-Jahresstatistik 2011/2010 (2012), http://www.bag.bund.de/SharedDocs/Downloads/DE/Statistik/Lkw-Maut/Jahrestab_11_10.pdf?_blob=publicationFile (accessed March 10, 2012)
[8] Bundesamt für Güterverkehr: Maut-Jahresstatistik 2012/2011 (2013), http://www.bag.bund.de/SharedDocs/Downloads/DE/Statistik/Lkw-Maut/Jahrestab_12_11.pdf?_blob=publicationFile (accessed June 10, 2013)
[9] Bundesministerium der Finanzen: Sollbericht 2013. Monatsbericht des BMF 2, 6–57 (2013), http://www.bundesfinanzministerium.de/Content/DE/Monatsberichte/2013/02/Downloads/monatsbericht_2013_02_deutsch.pdf?_blob=publicationFile&v=4
[10] CEN: ISO/TS 17575-1:2010 electronic fee collection - application interface definition for autonomous systems - part 1: Charging (2010)
[11] Coulouris, G.F., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design. Addison-Wesley (2011)
[12] Dettmar, M., Rottinger, F., Jestädt, T.: Achieving excellence in GNSS based tolling using the example of the german HGV tolling system. In: Proceedings of the 9th ITS Europe Congress (2013)
[13] Fishman, G.S.: Discrete-Event Simulation: Modeling, Programming and Analysis. Springer, Berlin (2001)
[14] Flötteröd, G.: Traffic state estimation with multi-agent simulations. Ph.D. thesis, TU Berlin (2008)
[15] Graham, S.L., Kessler, P.B., Mckusick, M.K.: Gprof: A call graph execution profiler. ACM Sigplan Notices 17(6), 120–126 (1982)
[16] Hericko, M., Hericko, M., Zivkovic, A.: An evaluation of different functional solutions for satellite-based tolling in europe. In: Hawaii International Conference on System Sciences, pp. 1–10 (2011), doi:10.1109/HICSS.2011.51
[17] Institute, E.S.: European user survey analysis. Report USV EUR 2.1 (1996)
[18] Intel: Intel VTune Amplifier, http://software.intel.com/en-us/intel-vtune-amplifier-xe (accessed May 12, 2013)
[19] Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers 100(1), 24–35 (1987)
[20] Lehman, M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE 68(9), 1060–1076 (1980), doi:10.1109/PROC.1980.11805
[21] Lehman, M.M.: The role and impact of assumptions in software development, maintenance and evolution. In: IEEE International Workshop on Software Evolvability, pp. 3–14 (2005), doi:10.1109/IWSE.2005.14
[22] Lunde, K., Kieble, L.: Simulating communication within a satellite-based automated toll collection system. In: Proceedings of the 55th International Scientific Colloquium, pp. 318–323 (2010)
[23] Lunde, K., Kieble, L., Funk, M.A.: Prediction strategies in a service level granting prefetching cache for version-controlled gis data. ISAST Transactions on Computers and Intelligent Systems 2(2), 46–51 (2010)
[24] MLDesign Technologies, Inc.: MLDesigner (2012), http://www.mldesigner.com/ (accessed April 10, 2012)
[25] Mosekilde, E.: Topics in Nonlinear Dynamics: Applications to Physics, Biology and Economic Systems. World Scientific Pub. Co. Inc., Singapore (1996)

[26] Opitz, F., Pfitzinger, B., Jestädt, T.: Service levels of a cost center organization. In: Alt, R., Fähnrich, K.P., Franczyk, B. (eds.) Practitioner Track International Symposium on Services Science (ISSS 2009), vol. 16, pp. 81–86 (2009)
[27] Pfitzinger, B., Baumann, T., Jestädt, T.: Analysis and evaluation of the german toll system using a holistic executable specification. In: 45th Hawaii International Conference on System Sciences (HICSS), pp. 5632–5638 (2012), doi:10.1109/HICSS.2012.111
[28] Pfitzinger, B., Baumann, T., Jestädt, T.: Network resource usage of the german toll system: Lessons from a realistic simulation model. In: 46th Hawaii International Conference on System Sciences (HICSS), pp. 5115–5122. IEEE (2013), doi:10.1109/HICSS.2013.415
[29] Pfitzinger, B., Bley, H., Jestädt, T.: Service catalogue and service sourcing. In: Abramowicz, W., Alt, R., Fähnrich, K.P., Franczyk, B., Maciaszek, L.A. (eds.) Informatik 2010, Business Process and Service Science–Proceedings of ISSS and BPSC, vol. 177, pp. 55–62 (2010)
[30] Pfitzinger, B., Gründer, T., Jestädt, T.: Sourcing decisions and IT service management. In: Alt, R., Fähnrich, K.P., Franczyk, B. (eds.) Practitioner Track International Symposium on Services Science (ISSS 2009), vol. 16, pp. 71–80 (2009)
[31] Pfitzinger, B., Jestädt, T.: Exploring the HGV fleet behavior: Notes from the German toll system. In: Proceedings of the 9th ITS Europe Congress (2013)
[32] Pfitzinger, B., Jestädt, T., Baumann, T.: Simulating the German toll system: Choosing 'good enough' driving patterns. In: für Verkehrstechnik, L. (ed.) Proceedings of the mobil.TUM 2013 – International Conference on Mobility and Transport. Technische Universität, München (2013)
[33] Pfitzinger, B., Jestädt, T., Helmers, W., Kosterski, S.: Best practices im contract lifecycle. In: Auerbach, M., Oecking, C., Jahnke, R., Strecker, F., Weber, M. (eds.) Best Practices im Outsourcing, pp. 185–200. Bitkom (2010)
[34] Ronngren, R., Barriga, L., Ayani, R.: An incremental benchmark suite for performance tuning of parallel discrete event simulation. In: Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, vol. 1, pp. 373–382 (1996), doi:10.1109/HICSS.1996.495484
[35] Tewoldeberhan, T., Verbraeck, A., Valentin, E., Bardonnet, G.: An evaluation and selection methodology for discrete-event simulation software. In: Proceedings of the Winter Simulation Conference, 2002, vol. 1, pp. 67–75 (2002), doi:10.1109/WSC.2002.1172870
[36] Toll Collect GmbH: Truck toll system proven effective (2012), http://www.toll-collect.de/en/company/news/truck-toll-system-proven-effective.html (accessed March 11, 2012)