Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1335–1341

DOI: 10.15439/2015F226
ACSIS, Vol. 5

Enhanced simulation performance through parallelization using a synthetic and a real-world simulation model

Tommy Baumann*¶, Bernd Pfitzinger‡§ , Dragan Macos†, Thomas Jestädt‡
*Andato GmbH & Co. KG, Ehrenbergstraße 11, 98693 Ilmenau, Germany. tommy.baumann@andato.com
¶Hochschule Aalen – Technik und Wirtschaft, Beethovenstraße 1, D73430 Aalen.
‡Toll Collect GmbH, Linkstraße 4, 10785 Berlin, Germany. {bernd.pfitzinger|thomas.jestaedt}@toll-collect.de
§FOM Hochschule für Oekonomie & Management, Zeltnerstraße 19, 90443 Nürnberg, Germany.
†Beuth Hochschule für Technik Berlin, Luxemburger Str. 10, 13353 Berlin, Germany. dmacos@beuth-hochschule.de

***Abstract*—Taking an existing large-scale simulation model of the German toll system we identify possibilities for parallelization in order to enhance simulation performance. We transform parts of the model from its current serial implementation to a parallel implementation. Afterwards we evaluate the achieved performance enhancement and compare the results to a synthetic benchmark model.**

I. Introduction

As technology advances in electronics, systems and processes with higher complexity, interconnectedness and heterogeneity can be developed. Simultaneously, user requirements are constantly increasing. Unfortunately "more is different" [1] as it is summarized in the definition of a "distributed system" as "one in which the failure of a computer you didn't even know existed can render your own computer unusable" [2]. Modeling and simulation techniques are applied to design, analyze, evaluate, validate, and optimize such complex systems. Especially in specification and design stage executable models deliver tremendous value by lowering the system design uncertainty – even expert advice is known to be over-confident [3], a well-known cognitive bias that needs to be mitigated by the system design process. Yet in many parts of everyday life people depend [4] on software-intensive systems.

The use of simulation models is one way to increase the specification speed and quality [5]. Hence, current system design approaches like Simulation Driven Design [6] are characterized by applying executable models to a large extend. A prerequisite to apply executable models is a so called execution domain: In our context Discrete Event Simulation (DES) [7] has gained significance. 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 (e.g. [8]), 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 long-term 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) [9]. In this context Parallel Discrete Event Simulation (PDES) [10] helps to provide the necessary simulation performance. In the article we identify possibilities for parallelization of a large-scale simulation model of the German toll system implemented in MSArchitect [11]. We transform parts of the model from the current serial, nonparallel implementation to a parallel implementation. Afterwards we evaluate the achieved performance enhancement.

The outline of the article is as follows: Section II gives an overview of the automatic German toll system and the corresponding simulation model. Typical use cases involve simulations at a scale of 1:1 spanning time periods of up to one year – necessitating a high-performance simulation model. The aim of this article is to investigate parallelized simulation models. To that end section III introduces the simulation framework architecture followed in section IV by a synthetic benchmark model for the use in PDES simulation. Section V describes the parallelized simulation model of the German automatic toll system and evaluates the simulation performance achieved. Section VI summarizes the results and describes future work and applications of our simulation model.

II. Simulation model of the German toll system

The German automatic toll system is a typical example of a state-of-the-art tolling system [12, 13] based on global navigational satellite systems (GNSS). At present it is the largest system of its kind in operations – collecting more than 4.3 bn € annually [14–16]. It was the first large-scale GNSS-based tolling system with more than 800 000 on-board-units (OBUs) deployed at present. More than 90% of the tolls are

978-83-60810-66-8/$25.00

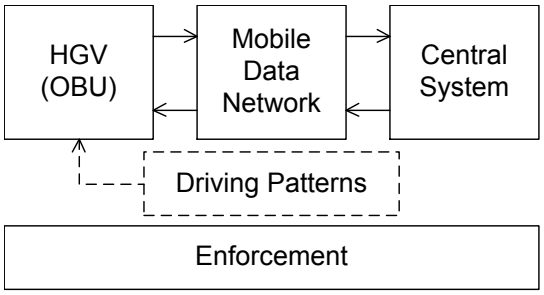

Fig. 1. High-level system design of a GNSS-based electronic tolling system and its dependency on the user interaction (driving patterns).

collected automatically the remainder using a manual process via internet or one of around 3 500 toll-station terminals.

The generic architecture of a GNSS-based tolling system is given in figure 1: The tolls are collected via OBUs installed in the heavy goods vehicles (HGVs) and transmitted via mobile data networks to the central system for processing. A separate part of the toll system is responsible for the enforcement and in the case of the German toll system an additional manual mode of tolling is implemented (not shown in the figure). The simulation model implements those technical systems and processes relevant for collecting tolls and for deploying updates to the OBUs: The vehicle fleet with its OBUs, the central systems required – i.e. a typical DMZ and the servers to receive toll data or provide updates – and the IP-based communication via mobile data networks.

Modeling the toll system we aimed to include all relevant processes at time scales of one second or longer, the technical systems in turn generate events with a higher temporal resolution in the model. However, the model of the tolling system needs to be accompanied by a model of the user behavior ("driving patterns" in figure 1) – the points in time when a given HGV is powered on or off, creates a toll event, looses or recovers its connection to the mobile data network.

The simulation model is applied in forecasting the dynamic behavior of the real-world tolling system – either the operations of the existing system (see section II-C) or for anticipated changes to the system (see sections II-A and II-B). Each application requires the simulation to predict the system behavior over several months – where the typical work-flow expects the simulation results to be available on the next business day. The existing simulation model is implemented to utilize a single CPU core and considerable effort has been expended to achieve an adequate performance [17–19]. However, over time the level of detail included in the simulation model and the number of OBUs in the real-world system increased. In addition simulation runs should deliver medium- and long-term predictions encompassing six months to one year.

A. Simulations in the Analyze and Design Phase

In the analysis phase we use the simulation model to validate the system requirements. Addressing the large amount of requirements in typical software-intensive systems, requirements are defined at different levels of abstraction: The top level defines why the system is built and what the owning organization hopes to achieve. This type is termed as business or stakeholder requirements. Already at this level-of-abstraction the requirements need to be validated as soon as possible – is the requirement really necessary at the documented level? Seemingly inconsequential numerical targets can have profound effects on the technical solutions, e.g. by necessitating a high-availability architecture.

The translation of the requirements into an executable specification (or simulation model) allows exploring the effects of the requirements on the solution space early on and vice versa: The virtual prototype transports operational properties of the real-world system back to the solution space potentially modifying or restricting the requirements. An example of our approach is the comparison of a thin-client and a thick-client solution in the domain of electronic tolling [20].

Our focus in this phase is to avoid system operation faults based on wrong assumptions or conjectures of the new requirements. The most important aspects checked by us are:

- Excessing the capacity limits of the key subsystems
- Appearance of non-valid system states
- The worst case scenarios in case of eventual system failures.

The aspects we want to simulate are designed manually. The most frequent checks are based on the system behavior checking concerning various system parameter values.

In the design phase the system architecture becomes more detailed: The analysis model is linked with the used frameworks, libraries and other third party software components such as database or GUI. The executable specification helps in drafting accurate requirements and the simulation runs yield the resulting dynamic behavior prior to the implementation of the system. At the same time, the simulation model becomes in itself more specific by adding the necessary behavior and parameters to allow measuring its performance. Depending on the complexity and runtime the optimization can be delegated to an automatic optimization algorithm.

In the design phase we can validate the system behavior with improved functional granularity. During this phase the software development process invests in design documents – consistent, fine-grained and formal models start to show up in the documentation. In our approach we can take these models as a starting point and use model transformations to transform the architecture models into the appropriate simulation model.

B. Simulations in the Develop and Test Phase

The implementation phases shift the focus to the system under construction. Here the requirements are supposed to remain fixed and only minor adjustments need to be returned to the requirements repository. The simulation model is an executable representation of the state-of-knowledge and is technically able to integrate a given component into the overall system – especially as long as the whole system is not yet available.

In that way simulations are part of the decision making process: Implementation variants can be explored and com-

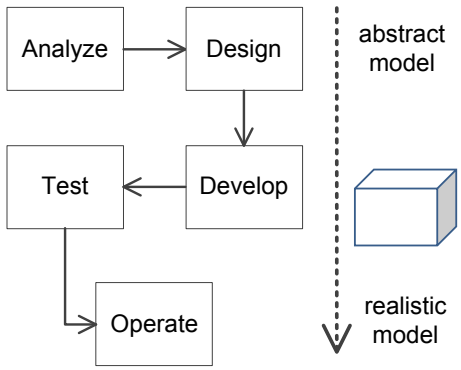

Fig. 2. The system development process (left) can be accompanied by a simulation model (right)

pared through simulations. The developed components are functionally integrated (transformed) and evaluated in the simulation environment. For an example one could look at the avionics industry [5, 8, 21].

In the test phase, the high-level requirements are used for acceptance tests of the whole system. Usually the development of the virtual prototype is considerably ahead of the real system. Therefore the soft- and hardware components are initially integrated into a whole working simulated system rather than the real world components: So called Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) tests. The simulation model provides the still missing components and allows to test dynamic coupling effects even when not all components of the real system are available. In other words we use the simulation environment as a stimulus for testing the individual software components – the simulation environment is a test driver for the developed software components. The test drivers are currently generated manually. Our goal is to generate them fully automatically via UML-enhancements to define the simulation test drivers during the software components design.

C. Simulations supporting System Operations

In the case of the German toll system the main objective of the simulation model is to support and to safeguard the day-to-day operations of the automatic toll system. To that extent the simulation model includes the most important processes (receiving toll data from the OBUs and sending updates to the OBUs) in a realistic way at a scale of 1:1. The dynamic behavior – with a strong daily and weekly rhythm – the predictions of the simulation model can and must be verified against data from the real-world system [22, 23] and typically show a good correlation [24].

Simulation models of the kind discussed here address the operator of a software-intensive system-of-systems. Typically the operator faces two challenges simultaneously: The day-to-day operations and the necessary changes and updates within the technical systems. Of course, the handling of technical details can and typically will be outsourced to specialized providers. The one challenge that remains for the system operator to handle is the integration of all technical and organizational parts into a working whole [25].

As in the test-driven-development (TDD) case, testing is not the aim of simulation-driven-development (SDD) rather the "driven [...] focuses on how TDD leads analysis, design, and programming decisions" [26]. In that sense, SDD tries to put the design to the ultimate test-case – the real-world operational context. The simulation model – an executable specification of the existing real-world system – is the starting point to focus any software development on the operational consequences. These consequences might be of a purely technical nature, e.g. the system architecture and performance, or include non-functional requirements and business or financial aspects. In particular these challenges dominate environments that are rich in legacy systems. The on-going development of theses systems is largely faced with integration issues between systems [27]. SDD addresses integration of systems as a cross-cutting concern by providing the software developer (or requirements engineer) with an executable copy of the real-world system.

III. Simulator Architecture and Benchmark Model

To transition from a sequential DES simulation model to a parallel one we first look at the possibilities offered by the simulation and modeling tool in use (see section III-A) and different ways of parallelizing existing models as discussed in literature (section III-B). The section ends with brief remarks on the PDES performance in general – the next section introduces and discusses a particular benchmark model.

A. Simulation and modeling tool, DES and PDES performance considerations

To model and simulate the structural and behavioral properties of the German toll system we selected and applied a system design tool for modeling and executing DES models [11]. The tool is specialized in integrated design of complex distributed systems and processes across different design levels. It offers a unique blend of performance [28] and customizability to manage extremely complex models within diverse usage environments. The simulation tool consists of several separated, mostly platform independent components, as a graphical user interface including a multi model editor, a simulation kernel, a library of standard model components, and a mission controller to run multiple simulations in parallel.

The simulation kernel used supports the execution of sequential DES and parallel DES (PDES) models. With DES and PDES the operation of a system is expressed as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur. Thus the simulation can directly jump in time from one event to the next. All events are managed by a so called future event list (FEL).

B. How to approach parallelizing DES models

PDES can substantially improve the performance and capacity of simulation, allowing the study of larger, more detailed

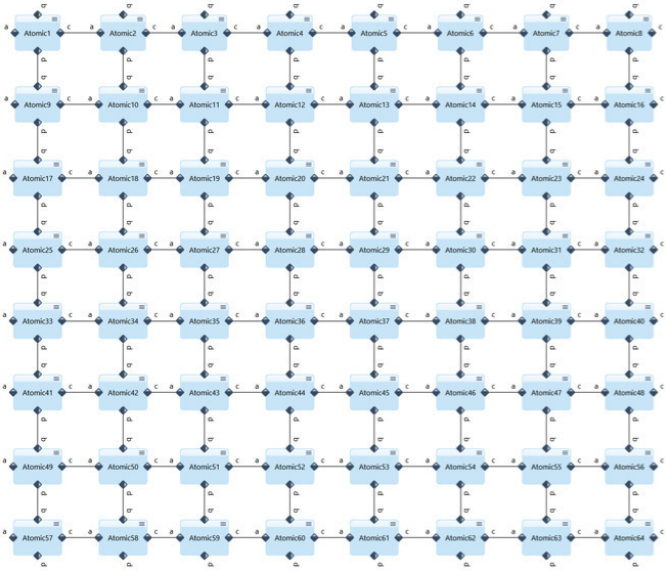

Fig. 3. Block diagram with a synthetic benchmark model using 8x8 network nodes.

models, in less time, and to be able to scale a problem if necessary. Thereby the performance and scalability is limited by communication latencies between the participating cores and nodes. Unfortunately, a prerequisite of parallelization is the decomposition of the model for processing on multiple processors or processor cores – it might become necessary to refactor existing serial mode models. This can be done in several ways [29]:

- Through the use of parallelizing compilers,
- In the form of replicated trials (run multiple serial simulations in parallel) or
- Distributed functions or
- Distributed events (with centralized FEL).
- Especially for optimizations a simple approach is the time-parallel domain decomposition (run multiple serial simulation in sequence) whereas
- the space-parallel domain decomposition (run multiple model parts in parallel) is used to accelerate a single simulation run.

The last enumerated decomposition approach, the space-parallel decomposition, is used by our simulation tool. Here, the simulation model is decomposed into sub-models or components. Each component is assigned to a process, where several processes may run on the same processor. This approach is applicable to any model and shows the greatest potential in offering scalable performance for complex models [30, 31]. Since the FEL is also decomposed into individual local FELs, it would never become the bottleneck. A higher degree of parallelism is expected because concurrent processing is encouraged.

In general, PDES approaches can be divided into two categories — conservative and optimistic — according to the way they handle the causality constraint of local FELs. Violating the causality constraint means that the future can affect the past leading to incorrect simulation results. In our case the simulation tool supports an optimistic synchronization mechanism, also known as time warp [32]. Causality errors are detected and fixed using rollback algorithms at the additional cost of necessitating rollback functions within the model.

C. PDES performance

Optimistic synchronization mechanisms have inherently more overhead than conservative synchronization mechanisms. The overhead includes e.g. state saving, global virtual time calculation and rollback steps. The degree to which they may affect the simulation performance depends among other things on the granularity of the model to be simulated and the support from the hardware. Both approaches – conservative and optimistic – have their advantages depending on the specific application case. In terms of general purpose simulation the optimistic approach appears to be slightly in advantage [33].

IV. PDES Benchmark model

In this section we introduce a synthetic network model consisting of a grid of simple networking units (section IV-A) and give the sequential and parallel simulation performance for different model sizes in section IV-B.

A. Description of the synthetic model

To evaluate the potential of parallelization we developed a synthetic benchmark model using our modeling and simulation tool. The model consists of several communication network nodes. When a network node receives an event, it consumes this event, handles some synthetic workload and sends a new event to a randomly selected neighbor network node with random delay. The synthetic workload on every event is approximately 10 000 floating point operations and the random delay is exponentially distributed with a mean of 5. Initially

TABLE I
Computation time in msec

	Threads	64 Nodes	128 Nodes	256 Nodes
Serial	1	8922	8953	8953
Parallel	2	5344	5109	4953
	3	3953	3719	3500
	4	3218	2985	2734
	5	2828	2532	2313
	6	2531	2235	2015
	7	2313	2016	1782
	8	2172	1875	1641

TABLE II
Speedup factor

	Threads	64 Nodes	128 Nodes	256 Nodes
Serial	1	1	1	1
Parallel	2	1,67	1,75	1,81
	3	2,26	2,41	2,56
	4	2,77	3	3,27
	5	3,15	3,54	3,87
	6	3,53	4,01	4,44
	7	3,86	4,44	5,02
	8	4,11	4,77	5,46

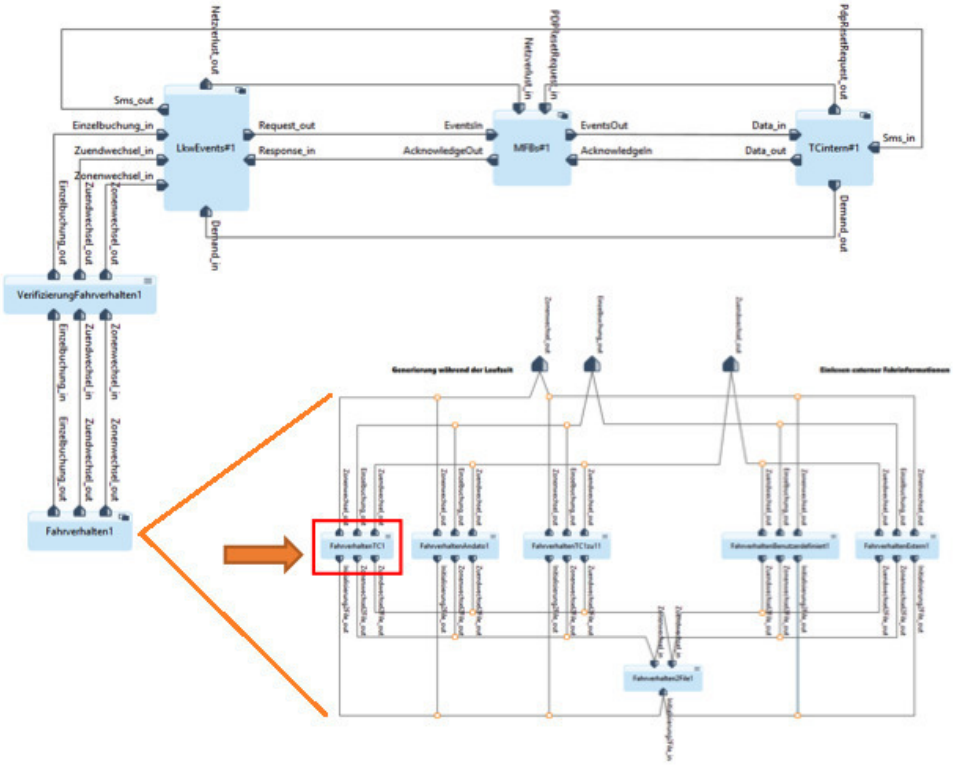

Fig. 4. Component responsible for generation of driving patterns

every second network node sends an event, which results in an event density of 50 percent. During a simulation run the total number of events in the system remains constant, due to the similar parametrization and behavior of all network nodes. Figure 3 shows the block diagram of the model with 8x8 network nodes.

B. Parallel and serial performance of synthetic model

In order to evaluate the simulation performance we simulated three sizes of the network model (8x8, 8x16 and 16x16 network nodes) using 1 to 8 threads. The clustering of the model and the assignment of these clusters to threads was done automatically by the simulation kernel. All simulations were executed on a PC using a i7-3635QM CPU with 2.40GHz, 4 cores and 8 threads and the simulation kernel of MSArchitect® Enterprise in version 2.3 with 64Bit.

The results are summarized in tables I and II. The first table gives the computation time needed for the simulation runs and the second table lists the relative speedup factor for each parallel simulation compared to the serial simulation. In theory the optimal speedup factor could be equal to the number of threads resp. cores used. Practically this is not possible due to the communication and synchronization overhead between the threads and the fact that the CPU does not offer 8 threads of equal computational power. All in all a substantial performance increase of up to 5.46 with 8 threads can be observed – higher than the 4 CPU cores offered and lower than the 8 parallel threads offered by the CPU.

V. Parallelization of the Simulation Model

The synthetic benchmark model presented in the previous section allowed a sizable speed-up through the use of a PDES simulation kernel. In this section we look at the challenges when an existing simulation model is the starting point of the parallelization effort (section V-A). In particular, some parts of the model are intrinsically serial – as required from the particular application domain. Section V-B discusses the process of selecting appropriate parts of the simulation model for parallelization.

A. Partial transformation as proof of concept

After successfully evaluating the parallel simulation approach using a synthetic model and the promising performance potential we decided to shift from serial to parallel simulation for the model of the toll system. Since the parallelization is a time-consuming process, we pursue an incremental model transformation, starting with the performance critical parts. In a first step we looked to the application-level performance of our model of the German toll system to locate appropriate model components for parallelization. This was done using both the kernel logging capabilities of our simulation tool and an external profiling application [17, 18]. 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,

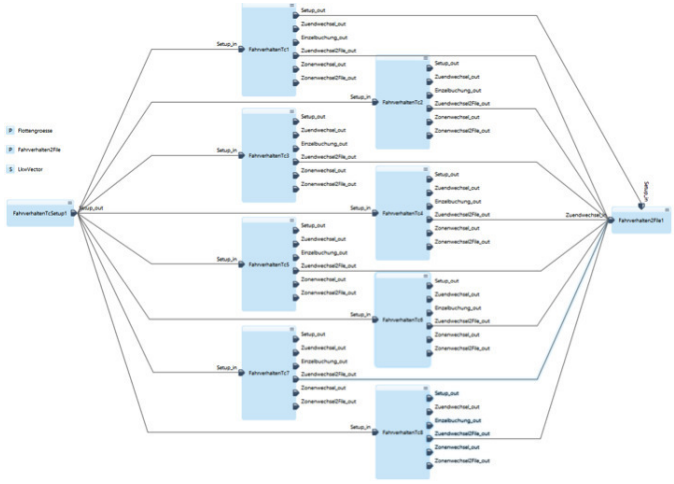

Fig. 5. Partial parallelization of the simulation model: Using 8 threads in generating the driving patterns.

CPU time), and the usage of particular instructions of a target program by collecting information on their execution.

To implement a partial parallelization we choose a setup using a similar hardware and software constellation as the synthetic model. The fleet size used in the simulation runs was set to 800 000 OBUs and a time frame of 20 weeks was considered. In the simulation runs the component responsible for generation of driving patterns was identified as performance critical (figure 4). This component consumes about 9.7% of the overall computation time in serial mode.

B. Selecting parts of the model for parallelization

To allow the execution of a given simulation model in parallel simulation mode the underlying PDES kernel calls a different set of atomic interface functions than in the serial mode: Instead of the function `run()`, which is called every time an event is received, the functions `runForward()`, `runReverse()` and `runCommit()` need to be implemented. This is necessary in the case of optimistic synchronization since this mechanism allows the speculative execution of events that might require a rollback to ensure causality. In addition the model architecture needs to be adapted following three rules:

- Reduce the dependencies between model components to simplify clustering,
- Transfer external states to internal states or to port based communication,
- Restructure the internal architecture of model components to enable functional parallelization, while leaving the interfaces stable.

In our case the selected model component is responsible for the generation of driving patterns for the whole vehicle fleet. We refactored this component so that the internal structures were multiplied according the number of parallel simulation threads (in our case 8) – the driving patterns of different heavy-goods-vehicles are inherently independent in our model. In that way each internal structure is responsible for one-eighth of the vehicle fleet, each executed as a separate thread (see figure 5). Executing the driving patterns component separately from the overall model in parallel mode let to a computation time of 3 min 30 sec compared to 8 min 28 sec in serial mode. This means a speedup of factor 2.4, which is quite good compared to the theoretical speedup factor of the synthetic model of factor 5.46.

C. Complete parallelization and backward compatibility

To parallelize the model of the German toll system completely the architecture of about 60% of the model components (composite blocks) need to be reworked according to the three architectural rules mentioned above. The remainder (mostly atomic blocks) need at least to be transferred to the set of parallel atomic interface functions. Both interfaces – for serial and parallel execution – can coexist in the simulation tool chosen for this work. This allows for an incremental porting of complex simulation models towards a fully parallelized model.

VI. Summary

Simulation models offer the ability to transform the software development process by placing each step into the realistic operational context. One important pre-condition is that the simulation model is sufficiently fast to generate realistic behavior at a scale of 1:1. Staying with a realistic level-of-abstraction the application of some models is limited by the execution time. In our example the large number of objects at run-time – models of on-board-units – and the necessity to predict the dynamic behavior of the system for a considerable length of time limit the use of simulation runs in the day-to-day decision making process.

In this paper we looked into increasing the simulation performance through parallelization either automatically or at the level of the simulation model. To further the discussion we implemented a synthetic network model with a configurable grid of identical components as a benchmark for the automatic parallelization achieved by the simulation kernel. In this case the speed-up compared favorably with the performance offered by the CPU chosen to run the benchmark.

In the real-world example the speed-up is more difficult to achieve – large parts of the model are either difficult to parallelize or contribute negligibly to the overall computation time. Profiling the simulation model we identified one part – the model of the user behavior – as both particularly time consuming and compact (as expressed by the size of this particular part of the model). Parallelizing only a part of the model limits the speed-up: The remaining model parts run in serial mode and we achieve only a speed-up of 2.4 using 8 threads. However, this speed-up is the result of re-factoring only 5% of the simulation model.

Future work is needed to automate the parallelization of existing models and to steer the modeling engineer to the most promising parts of a given simulation model.

REFERENCES

[1] Philip W. Anderson. More is different. *Science*, 177(4047):393–396, 1972. doi: 10.1142/9789812385123_others01.

[2] L. Lamport. Distribution, May 1987. URL http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt. [accessed 19-Mar-2015].

[3] Dale Griffin and Amos Tversky. The weighing of evidence and the determinants of confidence. *Cognitive psychology*, 24(3):411–435, 1992. doi: 00l0-0285/92$9.00.

[4] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, pages 11–33, 2004. doi: 10.1109/TDSC.2004.2.

[5] Tommy Baumann. *Automatisierung der frühen Entwurfsphasen verteilter Systeme*. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken, Germany, 2009. ISBN 978-3-8381-1266-4.

[6] Tommy Baumann. Simulation-driven design of distributed systems. *SAE Technical Paper*, 2011. doi: 10.4271/2011-01-0458.

[7] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24–35, 1987. doi: 10.1109/TC.1987.5009446.

[8] N. Fischer and H. Salzwedel. Validating avionics conceptual architectures with executable specifications. *Journal of Systemics, Cybernetics & Informatics*, 10(4), 2012.

[9] Bernd Pfitzinger, Tommy Baumann, and Thomas Jestädt. Network resource usage of the German toll system: Lessons from a realistic simulation model. *46th Hawaii International Conference on System Sciences (HICSS)*, pages 5115–5122, 2013. doi: 10.1109/HICSS.2013.415.

[10] Richard M. Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley-Interscience New York, 2000. ISBN 978-0471183839.

[11] Msarchitect. URL http://www.andato.com/. [accessed 10-Dec-2012].

[12] 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.

[13] Andrew T. W. Pickford and Philip T. Blythe. *Road user charging and electronic toll collection*. Artech House London, 2006. ISBN 978-1-58053-858-9.

[14] Bundesministerium der Finanzen. Haushaltsabschluss 2011, Feb. 2012. ISSN 1618-291X. URL www.bundesfinanzministerium.de/Content/DE/Monatsberichte/Publikationen_Migration/2012/02/inhalt/Monatsbericht-Februar-2012.pdf?__blob=publicationFile&v=3. [accessed 09-May-2012].

[15] Bundesministerium der Finanzen. Sollbericht 2013. *Monatsbericht des BMF*, (2):6–22, Feb. 2013. ISSN 1618-291X. URL http://www.bundesfinanzministerium.de/Content/DE/Monatsberichte/2013/02/Downloads/monatsbericht_2013_02_deutsch.pdf?__blob=publicationFile&v=4. [accessed 20-Mar-2013].

[16] Bundesministerium der Finanzen. Haushaltsabschluss 2013, Jan. 2014. ISSN 1618-291X. URL http://www.bundesfinanzministerium.de/Content/DE/Monatsberichte/2014/01/Downloads/monatsbericht_2014_01_deutsch.pdf?__blob=publicationFile&v=6. [accessed 26-Nov-2014].

[17] Tommy Baumann, Bernd Pfitzinger, and Thomas Jestädt. 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)*, pages 901–909. IEEE, 2012. ISBN 978-1-4673-0708-6.

[18] 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.

[19] Tommy Baumann, Bernd Pfitzinger, and Thomas Jestädt. Simulation driven development of the German toll system – simulation performance at the kernel and application level. In *Advances in Business ICT*, volume 257, pages 1–25. Springer International Publishing, 2014. doi: 10.1007/978-3-319-03677-9.

[20] 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.

[21] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. Airbus fly-by-wire: A total approach to dependability. In Renè Jacquart, editor, *Building the Information Society*, volume 156 of *IFIP International Federation for Information Processing*, pages 191–212. Springer US, 2004. ISBN 978-1-4020-8156-9. doi: 10.1007/978-1-4020-8157-6_18.

[22] Robert G. Sargent. Verification and validation of simulation models. In *Proceedings of the 37th conference on Winter simulation*, pages 130–143. Winter Simulation Conference, 2005.

[23] R.G. Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference (WSC)*, pages 166–183, Dec 2010. doi: 10.1109/WSC.2010.5679166.

[24] Bernd Pfitzinger, Dragan Macos, and Thomas Jestädt. Exploring the heavy goods vehicle fleet behaviour through simulations: Notes from the German toll system. *IET Intelligent Transport Systems*, Aug 2014. ISSN 1751-956X. doi: 10.1049/iet-its.2013.0175.

[25] 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.

[26] David Janzen and Hossein Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *Computer*, 38:43–50, Sep 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.314.

[27] Azad M. Madni and Michael Sievers. Systems integration: Key perspectives, experiences, and challenges. *Systems Engineering*, 17(1):37–51, 2014. ISSN 1520-6858. doi: 10.1002/sys.21249.

[28] A. Pacholik, T. Baumann, W. Fengler, and D. Grüner. Discrete event simulation performance – benchmarking simulators. In *International Simulation Multi-Conference (SummerSim)*, Genoa, Italy, 2012.

[29] Voon-Yee Vee and Wen-Jing Hsu. Parallel discrete event simulation: A survey.

[30] R. Righter and J.C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, Jan 1989. ISSN 0018-9219. doi: 10.1109/5.21073.

[31] A. J. Wing. Advances in parallel algorithms. chapter Discrete Event Simulation in Parallel, pages 179–226. John Wiley & Sons, Inc., New York, NY, USA, 1992. ISBN 0-470-21907-6.

[32] David Jefferson and Henry A Sowizral. Fast concurrent simulation using the time warp mechanism. 1982.

[33] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84545.