Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1125–1128
DOI: 10.15439/2016F33
ACSIS, Vol. 8. ISSN 2300-5963

# Applying simulations: On the importance of the simulation performance.

Bernd Pfitzinger*, Tommy Baumann‡§, Dragan Macos†, Thomas Jestdt*
*Toll Collect GmbH, Linkstrae 4, 10785 Berlin, Germany. {bernd.pfitzinger|thomas.jestaedt}@toll-collect.de
‡Andato GmbH & Co. KG, Ehrenbergstrae 11, 98693 Ilmenau, Germany. tommy.baumann@andato.com
§Hochschule Aalen – Technik und Wirtschaft, Beethovenstrae 1, D73430 Aalen.
†Beuth Hochschule fr Technik Berlin, Luxemburger Str. 10, 13353 Berlin, Germany. dmacos@beuth-hochschule.de

***Abstract*—Creating new software or software-intensive systems is still a challenge and far removed from a traditional engineering domain. The increasing size of software deployed in typical systems and the emergence of very large highly distributed systems necessitates additional techniques to assure the systems quality. Using the example of the German automatic toll system we briefly outline a simulation driven development approach: Using simulation models starting with the very early design stages to verify and validate the overall dynamic system behavior throughout the whole development process. In practice the approach depends particularly on the performance of the simulation model: Many simulation runs are necessary while exploring the solution space of a proposed change or while calibrating and optimizing parameters of the simulation models. Starting with an existing model of the German automatic toll system we look at two different possibilities for parallelization – parallelized optimization and the partial transformation of the simulation model to a parallelized implementation.**

## I. Introduction

MOST of the critical system design problems occur in the early design stages while specialists are specifying the system under a high degree of uncertainty. Many studies show that the probability of critical problems due to poor design decisions is very high in the specification phase [1]. The root cause is that either text-based or non-executable, model-based specifications are utilized: We build models of complex systems because we cannot comprehend any such system in its entirety [2]. Such specifications cannot be validated at the system level where the architecture and performance is determined as an emergent property.

The typical system development process will of course try to mitigate the consequences e.g. by shortening the step from the design to the deployment stage. But in distributed systems – "one in which the failure of a computer you didn't even know existed can render your own computer unusable" [3] – the properties of the whole system emerge only when all subsystems are integrated into a complete system.

Hence we propose to accompany the system development process with an executable specification of the whole system: At any stage during the system development process we implement the current level of detail as a simulation model. In that way the specification becomes executable (i.e. has sufficient detail to be executable).

In this article we briefly touch upon the concept of simulation driven development (SDD) – the starting point of an investigation into the performance of simulation models. When the intention is to base the system development process on simulation models the simulation results should be valid and readily available at any time. At least for complex systems both apsects necessitate a high simulation performance.

In particular the article applies two different ways of parallelizing simulations: The trivial parallelization of a genetic algorithm is of use when the simulation model is used either to explore the solution space in search of an optimal configuration or when parameters need to be fitted to data observed in the real world, e.g. modeling the user interaction. The non-trivial parallelization concerns the simulation model itself where – depending on the specific application – parts might be run independently. Starting with a simple benchmark model we apply the parallelization to an existing simulation model of the German automatic toll system.

The article is split into three parts: The first section explains the concept of simulation driven development in more detail (see section II) whereas the second and third section cover the simulation performance as one prerequisite for basing the software development process on simulation models: Parameter optimization using a genetic algorithm and the partial parallelization of a simulation model (sections III and IV). All parts have the same use case in common (section III): Applying the approach in the context of the German automatic toll system – a large scale liability-critical distributed system and a typical example of an electronic tolling system based on global navigation satellite systems (GNSS) [4].

## II. Simulation Driven Development

SDD is characterized by applying modeling and simulation technologies during the whole product lifecycle: An executable model exists at any time encapsulating the current knowledge of the system. The benefit of SDD lies both in the early design stages (when most of the important design decisions are made) and the ability to verify and validate the system at any time through executable models.

### A. Design approach

In general a system specification defines the functional and non-functional properties of a system in a formal, consistent, and self-contained manner to enable processing. Functional properties define the tasks of the system including information

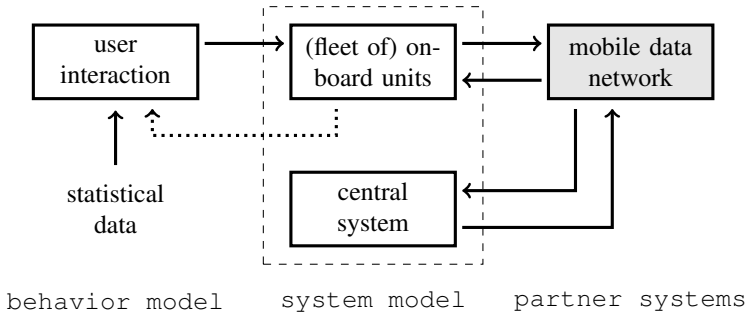


Figure 1. The simulation model of the German automatic toll system encompasses the the user interaction (left), the operators' technical systems (center) and systems under the responsibility of partners (right).

processing in relation to data, operation and the systems behavior.

Non-functional properties are more difficult to pin down there is not even a consensus on the term and its use [5]. They are used to describe the circumstances necessary to render the required functionality, e.g. the performance requirements, quality properties and constraints. In a sense the non-functional requirements become either constraints (to the system development or the system operations) or emergent properties (in the worst case emerging only at the system level when the user interaction is taken into account). SDD allows modeling both the constraints and the whole system including the user interaction and feedback loops.

Unfortunately, the specification phase is far removed from the final acceptance testing and the transition to operations. Many process models exist to close the gap, either by introducing more tests and documentation along the way or by shortening the development cycle (e.g. shifting requirements to the next cycle). To illustrate the system engineering process we take the "V-model" [6], a highly adaptable meta-model commissioned by the German federal government. At the core of this model is the well-known waterfall model dating back to 1970 with the design and development phases placed on the left and the test phases on the right – forming the letter "V". While it proceeds through the typical phases it differentiates the testing phase so that every design or development phase also sets up its own test phase.

Executable system specifications allow the validation, analysis, evaluation, and optimization of complex distributed systems even before the first line of source code is written – including dynamic interactions. Bottlenecks and resource shortages owing to dynamic coupling effects can be captured and resolved without running or implementing the real system. On a meta-level the executable model can include the system development process as well.

### *B. Applying SDD*

In its current implementation we used SDD in the specification phase of two upcoming changes to the application-level protocols governing the interaction between the on-board-unit (OBU) and the central systems of the German automatic toll system in essence switching from an open-loop controller (the OBU decides in large parts independently if and when to connect to the central system) to a closed-loop controller (where the control resides at any time with the central system). The impact of SDD on the specification phase was twofold: Performing simulations of the whole integrated system at a scale of 1:1 allows predicting the system load for the chosen system architecture (and in combination with a cost model the estimation of operational costs) in effect operational risks become visible in the very first step of the system development process.

So far, researching the impact of architectural choices (or the systems parametrization) is a manual task: The simulation model needs to be altered (or at least re-configured) and the simulation run performed and analyzed. Using a fitness-function to grade the simulation results (e.g. the correlation with the real-world update rates observed for a fleet-wide update) is the first step to automate the design process: An additional optimization algorithm is able to change the parametrization or even the systems architecture automatically and to search for the overall optimal solution.

An intangible benefit of SDD in our example is the emphasis on the operational performance of any change already during the specification phase: Using the simulation model the consequences to the day-to-day performance is quantified and easily comparable to todays (or the intended future) performance.

## III. Simulation model of the German toll system

In practice we introduced SDD to the system development process of Toll Collect GmbH (e.g. [7] and references therein) the operator of the German toll system for heavy goods vehicles (HGVs).

The automatic toll system consists of four major parts (figure 1): It uses OBUs to automatically collect the toll charges due. Most of the time the OBU is running independently and rarely connects to the central system via a mobile data network connection to upload the tolls collected and possibly download updates to its software, geo and tariff data. Using this architecture the reliability of the system depends heavily on the OBU its hardware, software and operational data as well as the user interaction.

In principle the system behavior could affect the user behavior (dotted line in figure 1): E.g. when the OBU signals 'out-of-order' the user could be tempted to quickly power-cycle the OBU. Particularly in the case of outages, e.g. when the central system cannot be reached by the OBU, the OBU will reach a point where it needs to signal the user a technical problem. It is therefore conceivable that the user behavior changes – either as intended by the system design (the user switches to the manual toll system, e.g. via internet booking) or by power cycling the OBU and thereby potentially triggering additional system load.

Looking at figure 1 it is interesting to note the first indications of a software-intensive system-of-systems: The system depends not only on the user interaction – it also includes partner systems that are operated independently (e.g. the mobile data network) and whose inner working is not accessible.

The model of the toll system depends on the external stimulus of the user interaction. With an emphasis on the fleet-wide propagation of updates we include only the temporal behavior: The points in time when an HGV is powered on (or off) and when a toll event is created.

Regarding the fleet-wide updates we use a genetic algorithm to adapt the simulation results to the data observed in the real-world system (for details and results see [8]). Parameter optimization requires the ability to compute a large number of results for different sets of parameters. In our case we used two sets of 16 probabilities (i.e. the probability for a given HGV of being active $N$ out of 16 weeks, once for German HGVs and again for foreign HGVs). Even using a running at a scale of less than 1:1000 a single simulation run takes almost one minute to complete.

## IV. Parallel DES models

For many purposes scaling is the appropriate solution – yet close to the specification limit of a system or once processes become non-linear, scaling is no longer an option. Simulation performance becomes important.

### A. Domain independent parallel DES models

Many approaches exist to parallelize existing serial mode simulation models [9]. In our case, the space-parallel domain decomposition accelerates a simulation run by executing parts of the model in parallel. This approach is applicable to any model and shows the greatest potential in offering scalable performance for complex models [10], [11].

An optimistic approach to parallelization is to execute components even if – at a later time – it becomes known, that the execution was unnecessary or violated a causality constraint ("time warp", [12]). As long as excess computing power is available the consequence is only that causality errors need to be rolled back, i.e. each component has to be enhanced by an additional rollback block and from time to time rollbacks will occur. In a real-world application the expected speed-up is limited by those parts of a domain-specific simulation model that are intrinsically serial.

### B. Domain specific parallel DES models

"Time warp" or optimistic parallelization has the advantage of being domain-independent – it is a feature offered by the simulation toolset as well as a programming paradigm. However, "artifical" rollback and commit steps need to be implemented and verified by additional testing. This effort could also be spent on an explicit, domain-specific parallelization of the simulation model itself.

In the example of the German automatic toll system (see figure 1) parallelism is inherent in many parts of the model (e.g. in the user behavior, the OBU fleet). Since most of the processing occurring in the simulation model is connected to the OBU fleet a sizable degree of parallelism could be achieved.

Simulation performance quickly becomes a bottleneck when the DES model is coupled to a real-world system, i.e. in a software-in-the-loop scenario. An example could be a server of the central system being subjected to the TCP/IP traffic generated by the OBU fleet.

Here the artificial concept of time – time is expressed in terms of events occurring and jumps immediately to the next event present in the "future event list" – needs to be coupled to the time passing in the real-world (system). Figure 2 depicts both concepts of time: In the real-world system time progresses continuously (indicated by the thick arrow from left to right) – aptly summarized as the wall-clock time. In contrast the DES tool considers time only in the presence of events: Each event generated in the simulation run carries a (future) time of execution and is accordingly put into the future vents list (FEL) for later processing. The current time in the DES model is therefore given by the pointer to the next event up for processing. Whenever multiple events occur at the same time a serial-mode simulation takes the events for processing from the future event list as if it were a queue. I. e. simultaneous events are processed sequentially while the clock is stopped.

When a real-world system is interfaced to the simulation model both possibilities break down: Even during times without events time passes at the same rate and sequential processing of simultaneous events will insert artificial delays, potentially disturbing the simulation results with two noteworthy consequences:

- Any speed-up of a DES simulation is negated once the real-world system is interfaced, the simulation will proceed (at most) at wall-clock time.
- The DES simulation model introduces artificial delays, when taking outgoing events from the FEL or inserting incoming events into the FEL – possibly to the degree of invalidating the simulation resuls by violating the 'real-time' constraint.

## V. Summary

As in the test-driven development (TDD) case, testing is not the aim of the SDD rather the "driven [...] focuses on how TDD leads analysis, design, and programming decisions" [13]. In that sense, SDD tries to put the design to

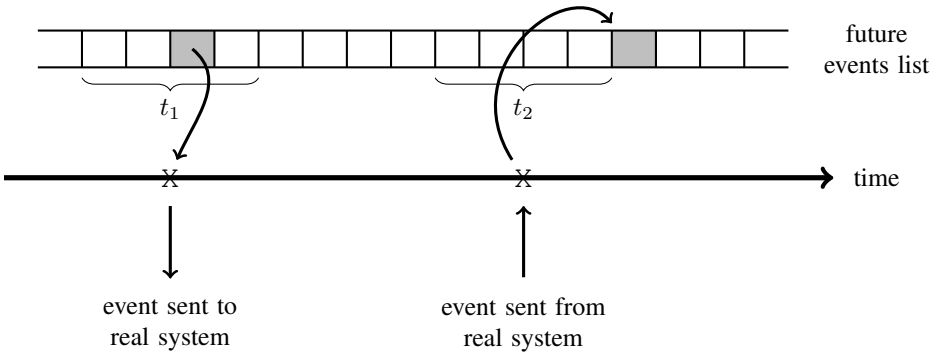


Figure 2. The concepts of time differ between the continuous time line in the real world and the list of (future) events at discrete points in time. Interfacing a real-world system will couple the two concepts.

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, where the on-going development is largely faced with integration issues. 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.

A prerequisite for applying SDD is the performance of the simulation model. We summarized two areas where performance matters: Exploring the solution space needs many simulation runs, albeit possibly at a small scale. Interfacing a simulation model with a real-world system is the final challenge – mingling simulated discrete and continuous real time.

## REFERENCES

[1] A. Aurum and C. Wohlin, in *Engineering and managing software requirements*, A. Aurum and C. Wohlin, Eds. Berlin: Springer, 2005, ch. Requirements Engineering: Setting the Context, pp. 1–15, ISBN: 978-3-540-28244-0. DOI: 10.1007/3-540-28244-0_1.

[2] ISO, *ISO/IEC 19505-2:2012 Information technology – Object Management Group Unified Modeling Language (OMG UML), Superstructure*. Berlin: Beuth Verlag, 2012.

[3] L. Lamport, *Distribution*, e-mail message, [accessed 16-Nov-2014], May 1987. [Online]. Available: http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt.

[4] J. Numrich, S. Ruja, and S. Vo, "Global Navigation Satellite System based tolling: State-of-the-art", *Netnomics: Economic research and electronic networking*, vol. 13, no. 2, pp. 93–123, Jul. 2012. DOI: 10.1007/s11066-013-9073-9.

[5] M. Glinz, "On non-functional requirements", in *15th IEEE international requirements engineering conference*, (Delhi), Oct. 2007, pp. 21–26, ISBN: 978-0-7695-2935-6. DOI: 10.1109/RE.2007.45.

[6] Verein zur Weiterentwicklung des V-Modell XT e.V. (Weit e.V.), *V-Modell XT version 2.0*, [accessed 21-Jan-16], 2006. [Online]. Available: http://www.v-modell-xt.de/.

[7] B. Pfitzinger, T. Baumann, D. Macos, and T. Jestdt, "Modeling regional reliability of 2G, 3G, and 4G mobile data networks and its effect on the German automatic tolling system", in *2015 48th hawaii international conference on system sciences (hicss)*, Jan. 2015, pp. 5439–5445. DOI: 10.1109/HICSS.2015.640.

[8] B. Pfitzinger, T. Baumann, D. Macos, and T. Jestdt, "Using parameter optimization to calibrate a model of user interaction", in *Proceedings of the 2014 federated conference on computer science and information systems*, M. P. M. Ganzha L. Maciaszek, Ed., ser. Annals of Computer Science and Information Systems, vol. 2, IEEE, Sep. 2014, pp. 1111–1116, ISBN: 978-83-60810-58-3. DOI: 10.15439/2014F123.

[9] V.-Y. Vee and W.-J. Hsu, "Parallel discrete event simulation: A survey", Tech. Rep., 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.7706.

[10] R. Righter and J. Walrand, "Distributed simulation of discrete event systems", *Proceedings of the ieee*, vol. 77, no. 1, pp. 99–113, Jan. 1989, ISSN: 0018-9219. DOI: 10.1109/5.21073.

[11] A. J. Wing, in *Advances in parallel algorithms*, L. Kronsj and D. Shumsheruddin, Eds. New York: John Wiley & Sons, Inc., 1992, ch. Discrete Event Simulation in Parallel, pp. 179–226, ISBN: 0-470-21907-6.

[12] D. Jefferson and H. Sowizral, *Fast concurrent simulation using the time warp mechanism: Part i, local control*. Santa Monica, CA: Rand Corporation, 1982.

[13] D. Janzen and H. Saiedian, "Test-Driven Development: Concepts, taxonomy, and future direction", *Computer*, vol. 38, no. 9, pp. 43–50, Sep. 2005, ISSN: 0018-9162. DOI: 10.1109/MC.2005.314.