On the Effectiveness of Communication-Centric Modelling of Complex Embedded Systems

Hugo Meyer*, Uraz Odyurt*, Simon Polstra*, Evangelos Paradas†,
Ignacio Gonzalez Alonso†, Andy D. Pimentel*
* University of Amsterdam, Amsterdam, The Netherlands
† ASML Netherlands B.V., Veldhoven, The Netherlands
*{*h.d.meyer, u.odyurt, s.polstra, a.d.pimentel*}*@uva.nl*, †{*evangelos.paradas, ignacio.alonso*}*@asml.com*

***Abstract*—Performance anomaly detection and prevention in complex industrial systems, involving many distributed embedded computing nodes, is an ever-present challenge. Understanding such complex systems using purely analytical, or experimental techniques is neither sufficient, nor cost efficient. Efficient high-level models that allow the study of current system behaviour, predict performance trends and assess possible optimisation measurements are in demand. This paper presents an approach to automatically infer such high-level system models by using system level tracing and to represent the current state of the system using Discrete Event Simulation (DES) techniques. Our approach allows us to follow the current behaviour pattern of the system by replaying observations, while at the same time, we are able to consider behavioural potentials under alternative sets of circumstances by exploring *what-if* questions. Our results are applicable to anomaly detection and anomaly prevention solutions for complex systems. The main approach in keeping our workflow as efficient and as compact as possible is the use of a communication-centric modelling approach of complex embedded systems. As our use-case, we discuss the main challenges of modelling software processes and resource utilisation in photolithography machines produced by ASML and how such complex systems can be mimicked with high-level event-based simulation. An automatically inferred OMNEST simulation model is presented and the different steps that were taken to simulate the behaviour of this production grade system are described. Initial evaluations show the maximum difference in application process lifetimes between real and simulated executions is less than 1%.**

***Index Terms*—System of systems, Modelling and simulation, Performance monitoring, Distributed Cyber-Physical Systems (DCPS)**

I. Introduction

The extreme complexity of industrial distributed embedded computing systems, Distributed Cyber-Physical Systems (DCPS) in particular, and any system of systems in general, results in highly dynamic behaviour [1]. This characteristic introduces challenges when optimising these systems to achieve better operational qualities and higher reliability [2], [3].

The process of optimising and improving a system's behaviour, which may be driven internally, or externally, involves different steps. As shown in Figure 1, the main steps are, *detection*, *imitation*, *prediction* and *enhancement*. The dynamic nature of complex systems dictates the need for continuous detection of behaviour. This is done by continuous monitoring of the system state and collecting different metrics, such as system latencies and resource utilisation. A model of the system under observation and the simulation based on it, mimicking the system's behaviour, will represent the imitation step. To keep this simulated behaviour accurately close to the actual system, the model needs to be calibrated using data from the detection step, in a continuous fashion. An accurate representation of the system enables prediction of possible scenarios, where the performance of the system can be negatively affected (performance anomalies). For instance, high CPU utilisation of some application processes could degrade the performance of others, in turn putting the whole system in a critical state. This involves resource utilisation trend analysis. Accurate predictions allow the usage of automatic, or manual actuation mechanisms, e.g., limiting the memory footprint of application processes, adapting their priority or changing their CPU scheduling policy. An important consideration is that the impact of an actuation should be correctly predicted and its impact should be monitored after deployment. In this paper, we focus on *detection* and *imitation* of complex system behaviour, which are the first two steps towards a complete solution for automatic anomaly detection and actuation. It is important to remark that throughout this work complex computing systems and DCPS are interchangeable terms. We focus on the *cyber* component of DCPS and not on addressing the physical aspects.

One of the many challenges to be addressed in order to optimise or steer system behaviour is efficient monitoring. The monitoring should capture sufficient behaviour, while abstracting it for simplification. Capturing fully detailed system behaviour will increase the complexity of the model, resulting in longer simulation times, which is not desirable. Accordingly, monitoring can be performed in two fashions. One can divide a complex system into its building blocks and subsystems and consider them as performance black-boxes, focusing on the inter-block communication. The alternative is to detect desired internal behaviour, invasively and by instrumenting the code itself. As will be agrued in this paper, focusing on certain subsystems, such as communication libraries promises a less laborious workflow, especially when aiming at automatic model inference.

This paper is composed as part of the research project 14208, titled *"Interactive DSL for Composable EFB Adaptation using Bi-simulation and Extrinsic Coordination (iDAPT)"*, funded by The Netherlands Organisation for Scientific Research (NWO).

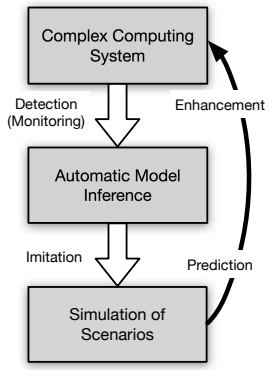


Fig. 1: Conceptual overview of the different steps in our model-based DCPS analysis and optimisation methodology

There can be numerous applications for our model-based DCPS analysis and optimization methodology. As a direct use-case, adaptive computing systems can benefit from it, for these systems are intended for reacting to changes, whether resulting from internal limitations, or the environment. The specific use-case we are focusing on is to analyse current and predict future system behaviour, in order to increase availability and scalability of the software running on a complex industrial system. By doing so, the availability and scalability of the complex industrial system itself will be increased. This need arises from the fact that within a distributed system, hundreds of processes compete for scarce resources. These resources might not be limited per se, if the hardware architecture provides them in abundance, but yet, software design could prevent efficient access and utilisation.

The main contribution of this paper is a methodology that focuses on mimicking the behaviour of an operational DCPS using high-level modelling and trace-driven simulation techniques. The feasibility of using a *communication-centric* tracing technique in combination with Discrete-Event Simulation (DES), to represent the full-system behaviour with sufficient accuracy, is discussed and analysed. As a use-case to evaluate our methodology, we have based our experimental validation on ASML photolithography systems, which are examples of complex DCPS. Initial results show that our communication-centric monitoring and modelling approach yields relatively accurate CPU utilisation trend estimates (with an average difference of around 10 percentage points of application process activity), while only capturing (i.e. , monitoring) the behaviour of 1/6th of all application processes. Simulations were executed using different event scheduling policies and the average total execution time difference between real and simulated processes is less than 1%.

The rest of this paper is organised as follows. In Section II we discuss related work and relevant state-of-the-art, while Section III elaborates our methodology. The results from our experiments are included in Section IV and Section V concludes the paper.

II. Related work

Modelling and simulation based analysis of distributed industrial systems is a well known notion and its challenges have been on the agenda of the scientific community [2], [3]. Accordingly, our high-level workflow involves *detection*, *imitation*, *prediction*, and *enhancement* phases, replacing the "design, collect information/data, build, execute, and analyse" problem solving cycle. Under our workflow, modelling and simulation steps fall under the imitation phase. As a starting step in understanding of embedded system behaviour and complex computing systems involving them, efforts in modelling and design-space exploration resulted in efficient offline methodologies [4] and tools, e.g., [5], [6].

High-level modelling and discrete-event simulation are widely used techniques in the domain of embedded systems. Most existing system-level embedded system modelling and simulation research [3]–[7] has focused on: 1) manually building (engineering) system models and 2) typically addressing the design of the embedded systems, not the online analysis of existing (already engineered) systems. Instrumenting software components of the system can be an effective way of collecting accurate and real-time information about systems' behaviour, as shown in [8].

The vast majority of embedded modelling and simulation studies focuses on relatively simple use-cases, such a multimedia systems [9]–[17]. Our technology demonstrator, as a production grade complex distributed system of systems, involves challenges beyond the ones mentioned in [2]. Previous research has looked at complex systems as a black-box [18], or examples such as [19] have considered process mining techniques [20]–[22]. These approaches use readily available offline information. In contrast, our work aims at an online solution for runtime modelling and simulation. Applications of data-driven model generation in other fields, such as hydroinformatics [23] and signal-transduction networks [24], also worth mentioning.

III. Performance analysis of complex systems

Complex computing systems are commonly composed of other subsystems that have been built separately and later on integrated together. As it has been mentioned, in this paper we focus on analysing the performance behaviour of such systems. We refer to performance as timing behaviour (latency, throughput, computation time etc.), as well as memory footprint of processes. In order to facilitate the performance analysis of complex DCPS and, eventually perform online management of the system behaviour, we firstly concentrated our efforts in designing efficient system monitoring techniques and high-level (application and hardware) models that accurately represent the performance behaviour of DCPS.

A. Methodology

Figure 2 depicts the main components of our methodology. As it can be observed, application processes are continuously monitored in order to feed a high-level application model that mimics the behaviour of the system with calibration data.

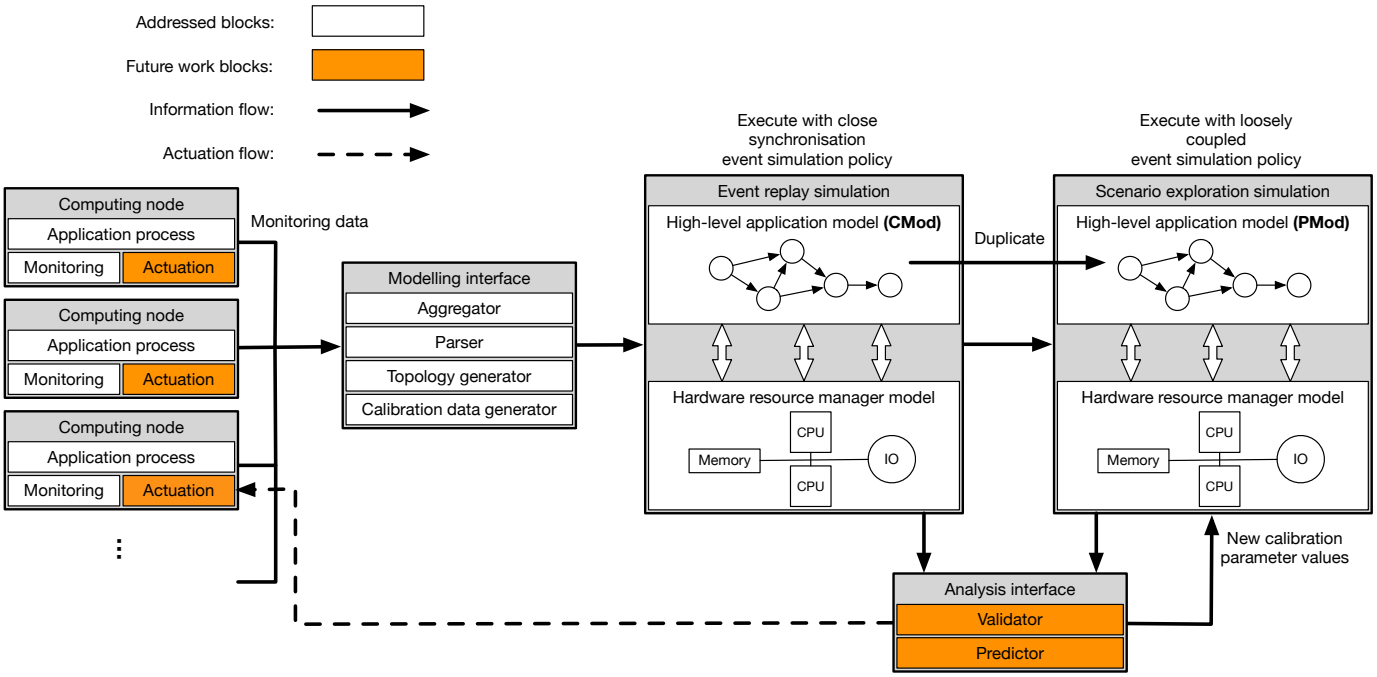


Fig. 2: Main components of the proposed solution methodology

This application model is called Current Model (CMod), for it represents the current state of the system. The CMod is used to detect trends in the usage of resources that can lead to performance anomalies in the form of machine hiccups, or full-system stops. Another high-level application model, called Prospective Model (PMod), initially derived from the CMod, constantly receives updates from it about the current status of the system. The PMod is intended to explore different scenarios, where performance behaviour can be managed and improved, resulting in actuation decisions over the monitored application processes. The analysis interface is in charge of the evaluation of trends, prediction of future scenarios and validation of actuation decisions. Actuation decisions could include changing CPU scheduling, or limiting per-process CPU usage, amongst others.

The time needed to design, collect information, build, and execute simulations can be reduced by using high-level simulation models that are calibrated using traces obtained from the system. Our system observation approach is based on lightweight software tracing techniques, applied to extract the behaviour of different software components of the system. Devising system observation points is critical in determining sources of bottlenecks and collecting performance metric values. Initial observations, or traces that our monitoring subsystem captures, are used to construct a system baseline, which in turn is used to calibrate the CMod. The CMod itself is automatically inferred from the monitoring data (details are covered in Section III-B3). Online calibration (using current system metrics) is a continuous activity, applied to CMod at runtime. In order to maintain a balance between the accuracy of simulations and the added overhead, we evaluate the *behavioural coverage* (the extent that the monitoring is capturing the real state of the system) of our observations and apply probing techniques with a reduced footprint.

In order to monitor the system behaviour, we use a communication-centric approach, which allows us to capture process behaviour by observing their interactions through a communication library. From this perspective, we identify tasks by observing communication and computation patterns of application processes and simulate how these processes compete for resources [1].

Figure 3 depicts our approach for modelling the system. Application process traces are converted to communication and communication events (in the form of read, write and compute events). Queueing models are used to represent the hardware and software components of the DCPS. Simulated processes send requests to the simulated components, and the service time in each of these queues is modelled as a function of the different parameters included in the request, such as CPU time, communication time, message size, among others.

Another important aspect that we take into account is the event scheduling policy applied to determine the order and timing of simulation events. Moreover, using an event simulation policy where the starting time of events is strictly dictated by the content of the traces will closely follow the current state of the system (appropriate for the CMod). However, if we want to evaluate or explore future scenarios, where system configurations can be different, the simulation events should not be scheduled to strictly follow the timing behaviour of collected traces, since a change in the configuration may affect the starting time of events, and in some cases the ordering. Relevant implementation details are covered in Section III-B.

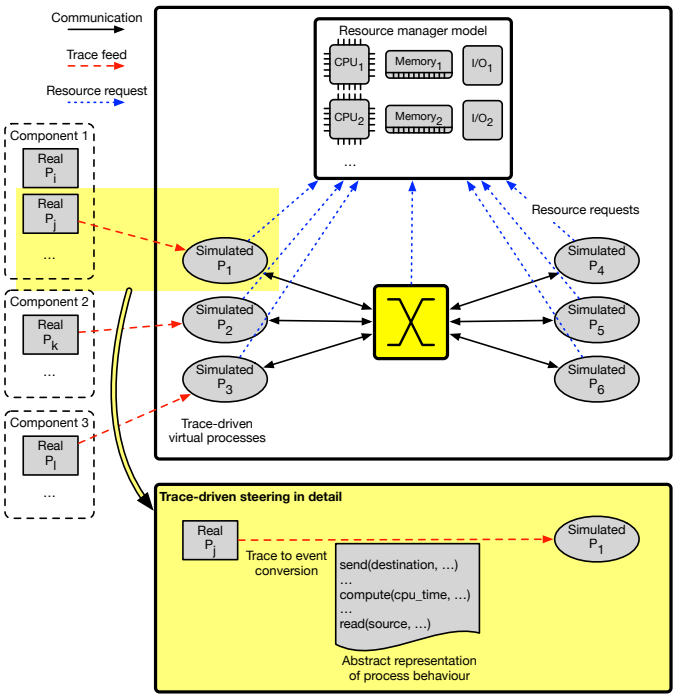


Fig. 3: Communication-centric modelling approach

B. From a methodology to an implementation

In this subsection we explain how the system monitoring module and the system simulation models were implemented. We have employed a communication-centric tracing technique and the OMNEST discrete-event simulation framework [25]. Queueing models are used to represent components of the DCPS and discrete-event simulation techniques allow us to analyse the status of the system during different phases of its execution. As shown before, simulations are trace-driven, where application processes are represented by traces (active entities) and system resources (passive entities, e.g., hardware, communication libraries) are represented by queues, where different service times are modelled.

In order to deploy the proposed methodology, the DCPS should be analysed to discover the most applicable approach for observing, modelling and simulating it, since different systems may have different observation points available and different requirements or metrics that should be taken into account. Also, proper tuning or actuation knobs should be determined depending on the main performance metrics that are considered in the context of the system. The main components of the implemented solution are described below.

1) Tracing application processes: As previously mentioned, to capture the trend of a system in an online fashion, we developed a tracing tool following a communication-centric approach, which allows us to limit the overhead introduced by invasive tracing. To capture the system behaviour, the monitoring module is implemented at the system communication module, i.e., the broker. When communication takes place, all processes using the monitored broker module shown in Figure 3 are indirectly traced. This approach reduces the amount of information captured, making the analysis of behavioural coverage (Section IV) extremely relevant. Pseudocode1 elaborates communication and computation event collection during the execution of processes. When a process starts its execution, a `trace_comp_start` call saves the *id* of the process, the *type of event* (`event`) and the *timestamp*. The process continues its normal execution until it calls the communication library, containing the tracing code (`communicate`). When entering the communication library, the `trace_comp_end` event saves the timestamp and the amount of resources that were used by the application until that point and the communication event is traced using `trace_comm_ini` and `trace_comm_end`. Then, collected traces are processed to calculate time spent in communication and computation related tasks. Computation events may contain not only CPU usage information, but also idle time, I/O time, etc., while communication events can also involve some CPU utilisation. In order to reduce the number of simulated processes, real process traces can be clustered into simulated processes according to their degree of similarity and frequency of operation. Different elements of the modelling interface from Figure 2 perform the following:

- Aggregator, gathers tracing information from different sources into a single log file,
- Parser, creates simulation events by converting traced action codes, such as write, reads, or computational events,
- Topology generator, analyses the trace information to automatically infer the interconnections and the number of application processes to be generated for the application model (using process IDs)
- Calibration data generator, provides the metric readings for the simulator and its trace-driven virtual processes, by analysing the raw traces.

Pseudocode 1: Communication-centric tracing

```
Function main():
    trace_comp_start (id, event,
     timestamp_start);
    do some work;
    communicate(payload1, dst1);
    do some work;
    communicate(payload2, dst2);
    ...
Function communicate(payload, dst):
    trace_comp_end(id, event, timestamp_end,
     CPU_used);
    trace_comm_ini(id, event, timestamp_start,
     dst, size(payload));
    communication occurs;
    trace_comm_end(id, event, timestamp_end,
     CPU_used);
    trace_comp_start(id, event, timestamp_start);
```

The outcome of tracing to be fed to the modelling interface, contains information such as, *event ID*, *process ID*, *event initiation timestamp*, *event end timestamp*, *event type*, *CPU utilisation*, and for communication events only, *message size* and *destination process ID*.

2) Hardware resource management: In the current version of our CMod implementation in OMNEST, the system hardware architecture is modelled as a module receiving requests from different simulated processes, storing the requests in a ready queue. CPU resource management is performed using a FCFS (First Come, First Served), or a Round Robin scheduling policy. CPU time is distributed as tokens that are given to the simulated processes and returned when the processing is finished. The number of tokens depends on the number of CPU cores included in the simulated system.

3) Trace-driven virtual processes: These are generic simulated processes, automatically generated based on the number of executed application processes in the real system, i.e., observed processes using communication-centric monitoring. Simulated processes are trace-driven and the simulation engine executes one event at a time. Process models are automatically inferred from the monitoring traces, avoiding the need to manually derive these models, which is typically very complex, or even infeasible to do for DCPS. Traces are processed by simulated processes and resource requests are created depending on the content of the trace, e.g., `CPU_used`, type of event (`event`) and timestamps. For instance, a process requiring a certain number of CPU cycles to perform the operation specified in the event, will create a CPU request. This request will arrive at the ready queue of the simulated CPU and will be served according to the CPU scheduling policy implemented in the resource manager model.

4) Event scheduling policies: As was mentioned, two simulation models are part of the proposed methodology, namely CMod and PMod. The CMod model should reflect, as closely as possible, the real behaviour of the system. It may actually be considered a model that replays collected traces. Figure 4 depicts our two investigated event scheduling policies. The simulated processes read the traces in order and one at a time and schedule the next event according to a number of factors considered in the event scheduling policies. These factors are, current simulation clock (t_{sim}), the initial timestamps of events encoded in the trace (t_{ini}), gaps between events in the trace (Δt), CPU utilisation and the idleness of events. In Figure 4, the t_{ini} is observed below each "Real trace" line and the t_{sim} is below each "Simulation" line. Figure 4a depicts our closely synchronised (replay) policy, *Policy 1*, with the first line showing two consecutive events from a real execution. When simulating events A and B, simulated event times (not the simulation itself) could be longer, or shorter. Whether event A finishes earlier or later, the next event's starting time will be its own t_{ini}, resulting in close synchronisation. The only exception is when A takes long enough to overtake B's t_{ini}. As such, B will start as soon as A ends,

$$t_{ini_B} = t_{end_A}.$$

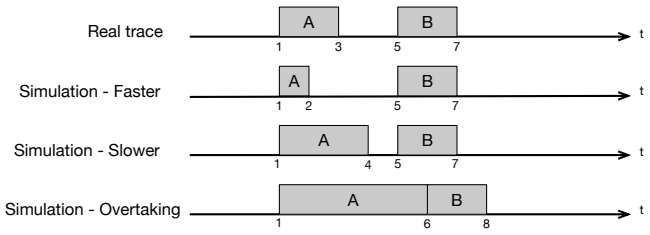
Figure 4b shows how gaps between events (Δt) and process idleness during an event are accounted for when scheduling *computation* events. *Policy 2* does not initiate events strictly based on their t_{ini}, but instead, here the focus is on gaps between events, such that

$$t_{ini_B} = t_{end_A} + \Delta t.$$

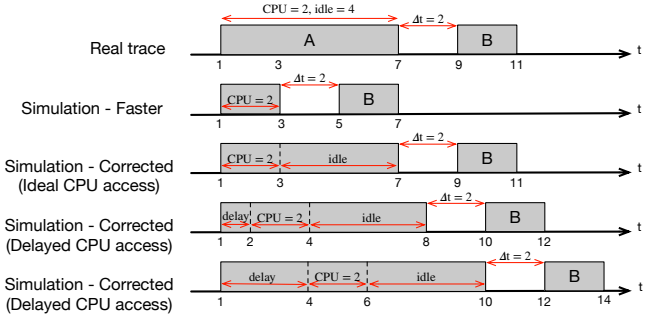
Another consideration is the difference between the duration of an event and its associated CPU time. In most cases, an event includes process idleness, which is always added to the simulated event in the form of a wait. This *idle* is added after the event's computation is over and the CPU token is released. However, CPU availability might not occur at the start, adding a delay to the event. Considering a delayed CPU access, the next event will start at

$$t_{ini_B} = t_{ini_A} + delay_A + comp_A + idle_A + \Delta t.$$

This second event scheduling policy focuses on increasing the flexibility when releasing events to the simulation engine by avoiding the usage of timestamps specified in the traces. While Policy 1 focuses on replaying the behaviour of the traces (suitable for CMod), Policy 2 tries to maintain the simulation accuracy as closely as possible to the real execution, without forcing events to be scheduled at specific timestamps. This makes the second policy more suitable to explore the impact of different actuation mechanisms (as intended in PMod).



(a) Policy 1: Close synchronisation of events based on traces



(b) Policy 2: Considering computation events' idleness when scheduling events

Fig. 4: Different event scheduling policies during simulation

C. Use-case: Photolithography systems

As noted in Section I, the above elaborated techniques have been applied to ASML's photolithography machines, as this

paper's real-world demonstrator. These complex DCPS include communication subsystems based on the *publish-subscribe architectural pattern*. The subsystem connects application processes from different software components, which in turn run on distributed computing nodes.

IV. Experimental results

We have assessed our aforementioned methodology using a simulation tool, developed with OMNEST and in combination with a communication-centric monitoring tool. The simulation tool follows the model described in Section III-B. The three main metrics that we have considered to evaluate the quality of our proposed methodology are: 1) the behavioural coverage of our communication-centric monitoring tool; 2) the accuracy of simulations regarding CPU utilisation and; 3) the accuracy of simulations considering process lifetimes.

In order to perform the simulations, traces from a fully functional ASML photolithography machine having the same software components as a production machine, and thus the same behaviour, minus all the moving robotic parts, were used. This machine is used by ASML to perform machine throughput qualification.

Traces were collected by introducing probes in one of the main communication libraries used in this system, which follows the broker architectural pattern. These traces were post-processed, and in turn, used as input for the simulation model developed in OMNEST. The collected traces correspond to four different workload scenarios[1] using a single recipe (pattern). In real operation, recipes are optically imaged onto a silicon wafer covered with a film of light-sensitive material. The load scenarios consist of: 1) printing the recipe onto 2 wafers, workload *2w*; 2) printing the recipe onto 10 wafers, workload *10w*; 3) sequentially printing the recipe onto a combination of 2-wafer and 10-wafer batches, workload *2+10w* and 4) sequentially printing the recipe onto a combination of 2-wafer, 5-wafer and 10-wafer batches, workload *2+5+10w*. The difference when processing different batches of wafers lies in the fact that different application processes are triggered, depending on the number of wafers to be processed, as well as the number of the batches.

Figure 5 shows how the CPU utilisation trend of the total system is closely matched with the combined CPU utilisation of the observed processes, i.e., the ones using the communication subsystem. We have compared both accumulated utilisation values captured via the UNIX '`top`' command and from our tracing events (collected via resource usage system-call, '`getrusage`'), with total system utilisation values from '`top`'. The absolute difference between CPU utilisation of the processes involved with the communication subsystem and total CPU utilisation of the full system represents the amount of undetected behaviour. The figure depicts the undetected behaviour with small amounts of dispersion (of which $0.7-8.7\%$ are outliers), indicating a matching behaviour throughout the execution time. As noted in Section I, the behaviour captured from the communication subsystem involves 1/6th of all application processes.

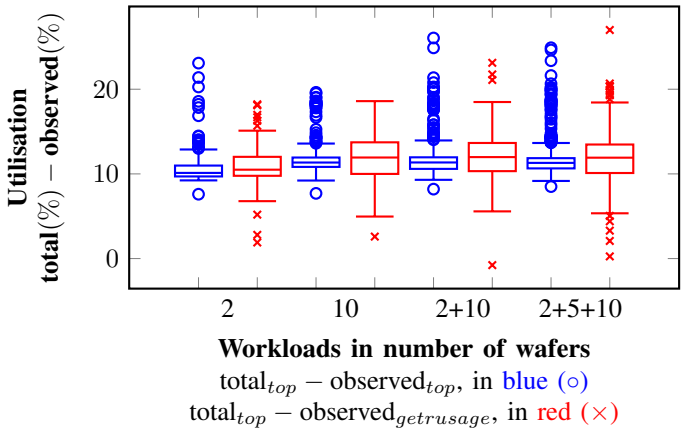


Fig. 5: Behavioural coverage at different stages: CPU utilisation differences for multiple workloads, total system vs. communication-centric view

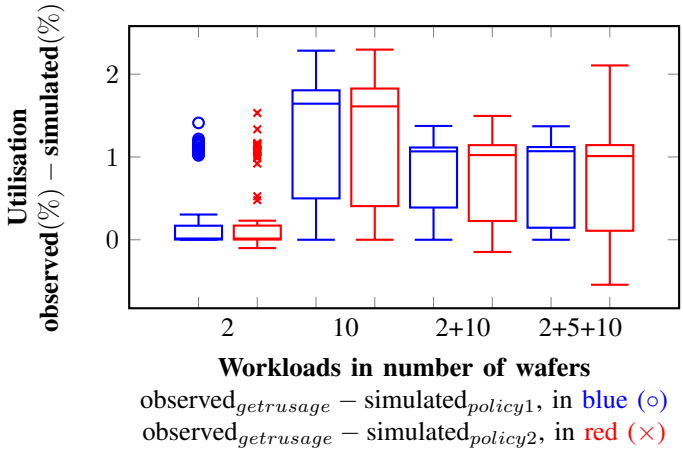


Fig. 6: CPU utilisation differences for multiple workloads, observed vs. simulated results

The blue box plot is based on a graph resulting from the absolute difference between full system CPU utilisation and observed CPU utilisation for every point in time, all from system parameters ('`top`'). It shows median values of 10.10%, 11.35%, 11.35% and 11.30%. The red box plot is generated similarly, with the only difference that observed CPU utilisations are based on recorded communication events data ('`getrusage`') and shows median values of 10.50%, 11.94%, 12.01% and 11.91%.

Figure 6 shows the absolute CPU utilisation difference obtained considering the two different simulation policies explained in the previous section. Median values for Policy 1 are 0.0149%, 1.6439%, 1.0684% and 1.0697%, while for Policy 2 these values are 0.0146%, 1.6119%, 1.0232% and 1.0108%. These results are achieved considering load sce-

[1]These workloads do not represent all possible workloads for photolithography machines, but are sufficient for our experiments.

narios 2w, 10w, 2+10w and 2+5+10w, respectively. For both simulation policies, the absolute CPU utilisation difference ranges between 0.5% and 2.3%, showing that the simulated CPU utilisation closely tracks the monitored CPU utilisation. Actual CPU utilisation trends are confidential and graphs depicting them cannot be presented.

Figure 7 depicts per-workload lifetime difference when simulating processes with the two previously covered event scheduling policies. The y-axis represents process lifetime difference for different workloads. The lifetime difference is calculated by subtracting the lifetime of an observed process (from the input traces) with a simulated process (from the output traces). Ideally, the lifetime difference should be as small as possible, indicating low deviation. Separate box plots are drawn for our workloads in Figure 7. There are specific processes where the lifetime difference can be considerable, such as the 221% lifetime difference observed in one of the processes, when using Policy 2 for the 2-wafer experiment. These major differences are present when processes have very short lifetime or contain one small set of events to simulate (in some cases only one event). Policy 2 presents, in most cases, a higher process lifetime difference, due to the fact that events are not scheduled according to timestamps recorded in the traces, and only idleness of computation events is taken into account, as shown in Figure 4b. For the case of Policy 1, some differences are higher because the last scheduled event can be a computation event, where the idleness is not considered. This may considerably increase lifetime difference.

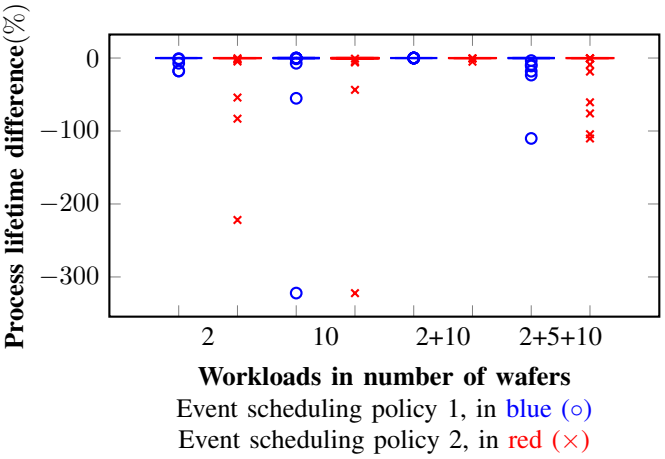


Fig. 7: Simulation of different workloads considering both Policy 1 and Policy 2

Even taking into account that some events have a high lifetime difference, the average total execution time difference, considering the two presented policies, ranges between -0.0007% (a negative value indicates underestimation) and 0. Simulating between 63 thousand and 480 thousand events, captured during 10 to 15 minutes of real machine execution, takes about 10 to 15 seconds (using a single core of an Intel® Xeon® Gold 6146, running at 3.20 GHz).

V. Conclusions and Future Work

Modern complex industrial systems present highly dynamic behaviour throughout their operational cycle, making the monitoring, trend analysis and automatic detection and resolution of performance anomalies very complex. In this paper, we have presented a methodology that focuses on online analysis and actuation of complex industrial systems by using a communication-centric modelling approach. The first components of a fully-automated solution to manage performance behaviour of the system, that entail system monitoring and high-level modelling, have been implemented and evaluated using a photolithography machine as a use-case.

Our experiments indicate that the communication-centric analysis of systems of systems, relying on communication subsystems is effective. Following this communication-centric perspective facilitates monitoring of the system at hand, resulting in a reduced collection of data, without an excessive loss in accuracy. The difference between captured and total CPU utilisation in our experiments is around 10 percentage points, while the difference between simulated (estimated) and captured CPU utilisation is between 0 to 2 percentage points. Furthermore, modelling, calibration, simulation and validation steps are facilitated as well, for these steps make use of the same compact (vs. monitoring a full system) data. We see this as an effective way of addressing the ever growing complexity.

Our future steps will focus on improving the accuracy of our communication-centric modelling approach by enhancing the behavioural coverage of the communication subsystem. We will also transition from post-execution processing of data to run-time by adapting our workflow to operate in an online fashion. Online representations of current and future states of complex systems are critical to enable automatic management.

References

[1] U. Odyurt, H. D. Meyer, S. Polstra, E. Paradas, I. G. Alonso, and A. D. Pimentel, "Work-in-progress: Communication-centric analysis of complex embedded computing systems," in *Proceedings of the 15th International Conference on Embedded Software*, ser. EMSOFT '18, 2018, short paper.

[2] J. W. Fowler and O. Rose, "Grand challenges in modeling and simulation of complex manufacturing systems," *SIMULATION*, vol. 80, no. 9, pp. 469–476, 2004.

[3] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, Jan. 2012.

[4] B. Kienhuis, E. F. Deprettere, P. van der Wolf, and K. A. Vissers, "A methodology to design programmable embedded systems - the y-chart approach," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, 2002, pp. 18–37.

[5] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, Feb. 2006.

[6] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 2–2, Jan. 2007.

[7] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 10, pp. 1517–1530, Oct. 2009.

[8] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10, Washington, DC, USA, 2010, pp. 79–88.

[9] E. Bondarev, P. de With, M. Chaudron, and J. Muskens, "Modelling of input-parameter dependency for performance predictions of component-based embedded systems," in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, ser. EUROMICRO '05, 2005, pp. 36–43.

[10] E. Bondarev, M. Chaudron, and P. H. N. de With, "Carat: A toolkit for design and performance analysis of component-based embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '07, Nice, France, 2007, pp. 1024–1029.

[11] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 299–310.

[12] H. Hoffmann, "Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems," in *Proceedings of the 2014 Agile Conference*, ser. AGILE '14, 2014, pp. 223–232.

[13] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Power optimization in embedded systems via feedback control of resource allocation," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 1, pp. 239–246, Jan. 2013.

[14] A. Muttreja, A. Raghunathan, S. Ravi, and N. K. Jha, "Automated energy/performance macromodeling of embedded software," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC '04, San Diego, CA, USA, 2004, pp. 99–102.

[15] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for mpsocs," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13, 2013, 131:1–131:6.

[16] M. Shafique, L. Bauer, and J. Henkel, "Enbudget: A run-time adaptive predictive energy-budgeting scheme for energy-aware motion estimation in h.264/mpeg-4 avc video encoder," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010, pp. 1725–1730.

[17] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, Jul. 2011, pp. 404–411.

[18] V. V. Parappurath, J. P. M. Voeten, and K. C. Kotterink, "Calibration error bound estimation in performance modeling," in *Proceedings of the 2013 Euromicro Conference on Digital System Design*, ser. DSD '13, 2013, pp. 97–102.

[19] A. Rozinat, I. S. M. De Jong, C. W. Günther, and W. M. P. Van Der Aalst, "Process mining applied to the test process of wafer scanners in asml," *Trans. Sys. Man Cyber Part C*, vol. 39, no. 4, pp. 474–479, Jul. 2009.

[20] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Advances in Database Technology — EDBT'98*, 1998, pp. 467–483.

[21] A. Weijters and W. van der Aalst, "Process mining discovering workflow models from event-based data," in *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, 2001, pp. 283–290.

[22] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[23] D. Solomatine, L. See, and R. Abrahart, "Data-driven modelling: Concepts, approaches and experiences," in *Practical Hydroinformatics: Computational Intelligence and Technological Developments in Water Applications*. 2008, pp. 17–30.

[24] K. A. Janes and M. B. Yaffe, "Data-driven modelling of signal-transduction networks," *Nature reviews Molecular cell biology*, vol. 7, no. 11, p. 820, 2006.

[25] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08, 2008, 60:1–60:10.