



Universiteit
Leiden

The Netherlands

Towards design space exploration of industrial distributed cyber-physical systems

Saadatmand, F.S.

Citation

Saadatmand, F. S. (2026, September 30). *Towards design space exploration of industrial distributed cyber-physical systems*. Retrieved from <https://hdl.handle.net/1887/4310251>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/4310251>

Note: To cite this publication please use the final published version (if applicable).

Chapter 1

Introduction

1.1 Context and Motivation

Cyber-Physical Systems (CPS) are systems that tightly integrate computational (cyber) components with physical processes. The cyber part consists of embedded software and computing hardware – often based on heterogeneous multi-core or many-core processors – responsible for control, data processing, monitoring, and decision-making. The physical part includes mechanical, electrical, or chemical components that interact with the environment through sensors and actuators. Together, these tightly coupled parts enable CPS to perform complex, adaptive tasks across application domains such as healthcare, industrial automation, robotics, avionics, and space.

In modern industrial applications, these systems often take the form of distributed CPS (dCPS), where multiple cyber-physical subsystems, each with its own computing and sensing capabilities, are networked together. Such distributed architecture enhances functionality, scalability, and system-wide coordination, but also introduces substantial architectural and behavioral complexity.

As the size and complexity of dCPS continue to grow, manufacturers face significant challenges in designing their next-generation machines based on dCPS. A key challenge is to evaluate how design decisions influence non-functional properties of dCPS such as performance, energy consumption, and cost. These evaluations must take place early in the design and development process, when critical architectural choices are still open. To support this, system designers need methods that can efficiently explore design alternatives and provide rapid answers to “what-if” questions about the system behavior under different configurations. This necessitates scalable and efficient system-level Design Space Exploration (DSE) methods tailored to the unique characteristics of industrial dCPS [107]. Such methods must integrate appropriate models of hardware and software behavior, efficient simulation and optimization techniques, and supporting tools for exploring a wide range of configurations.

Most of the existing and well established DSE methods have been developed for embedded Systems-on-Chip (SoC) and are not directly applicable to industrial dCPS due to several obstacles. First, these methods typically lack a global system model that can efficiently capture the behavior of distributed and heterogeneous components interconnected with a complex network infrastructure. Second, manually constructing such model is extremely difficult and often infeasible due to the scale and dynamic behavior of industrial dCPS. Third, even when suitable models can be constructed, often there is no scalable simulation and optimization engine that can efficiently explore large system-level design spaces.

The aforementioned limitations call for new methodologies specifically designed for industrial dCPS. These methodologies must combine efficient high-level system modeling, automated model derivation techniques, and scalable simulation tools to support meaningful and practical system-level evaluation and design space exploration.

1.2 Scope and Focus of This Thesis

To address the limitations discussed above, this thesis presents and develops a methodology for scalable and efficient DSE tailored to industrial dCPS. The focus is on enabling informed design-time decisions during the early development phase of the cyber part of dCPS, which plays a central role in shaping the system behavior and performance. As dCPS expand in size and variability, the design and optimization of their cyber components becomes increasingly complex, especially in the face of heterogeneous architectures and dynamic runtime conditions [6]. This complexity is further amplified by the trend toward software-intensive architectures, where more functionality is implemented in software to facilitate adaptability and faster system evolution [1]. As a result, the cyber part not only introduces many of the key design challenges but also offers the most leverage for system-level optimization, making it a strategic focus for DSE.

The proposed methodology in this thesis defines appropriate system-level models at a high level of abstraction, while preserving sufficient accuracy to support meaningful performance analysis and architectural trade-off evaluation. In addition to the model definitions, it incorporates techniques and tools for automatically deriving these models from run time data collected from existing dCPS and transforming them into executable representations. These representations form the foundation for scalable simulation and performance analysis.

To validate the methodology, it is applied to the TWINSKAN lithography system,

one of the advanced machines developed by ASML [108]. This system represents a complex industrial dCPS and provides a concrete basis for assessing the effectiveness of the proposed DSE methodology.

1.3 Scientific Challenges

To realize a practical and scalable DSE methodology for industrial dCPS, several scientific and technical challenges must be addressed. These challenges cover the entire methodology, from defining a conceptual modeling framework for system-level DSE to developing executable models, automation tools, and efficient simulation environments. In particular, DSE must be adapted to the nature of industrial dCPS, that are typically distributed, heterogeneous, and exhibit dynamic execution behavior. These characteristics introduce obstacles that cannot be adequately addressed by traditional embedded system design techniques.

A fundamental starting point is the construction of the overall DSE methodology. As indicated in Section 1.1, the methodology must combine system modeling, automated model derivation and transformation techniques, and system performance evaluation tools. This requirement leads to the first challenge:

CHL1: Devise a DSE methodology for the cyber part of complex industrial dCPS that integrates efficient system modeling, automated model derivation and transformation, and system performance evaluation into a coherent and scalable workflow.

A mature and effective DSE methodology for the cyber part of dCPS requires system modeling that encompasses three core models [50, 57]:

1. A **hardware platform model** describing the available computing infrastructure, such as processing components and interconnection networks;
2. An **application software model** describing the behavior of distributed software processes and their interactions;
3. A **mapping model** specifying the assignment of the software processes to the hardware processing components.

Taking into account the high complexity of industrial dCPS, the aforementioned

models must represent the system at a high level of abstraction in order to manage the complexity and enable efficient and scalable DSE.

Additionally, industrial dCPS typically operate in multiple operation modes. An operation mode represents a specific operational condition resulting from a particular combination of hardware and/or software configuration parameters that are externally imposed by the system setup or usage scenario. Different operation modes may lead to different execution behaviors, for example in terms of resource utilization, execution order, and interaction patterns between system components. Therefore, the system modeling must be able to capture multiple operation modes in an abstract way.

Moreover, to support model reuse during exploration of the application software mapped onto different dCPS hardware platform architectures, the application software model must remain independent from platform-specific artifacts. All these requirements lead to the second challenge:

CHL2: Define an application software model and a hardware platform model that are independent of each other and highly abstract, yet sufficiently accurate to support meaningful and practical performance analysis of dCPS in different operation modes.

Even with such models defined, manually constructing them for an industrial-scale dCPS is infeasible. Such dCPS often consists of hundreds of interacting software processes distributed across multiple computing nodes, with behavior that varies significantly across operation modes and configurations. Capturing this behavior accurately, while ensuring the modeling process remains scalable, is extremely challenging. In addition, utilizing techniques and tools for automated source code analysis to infer system behavior is often infeasible. Many industrial dCPS are composed of huge code-bases developed in multiple programming languages, making it extremely difficult, or even impossible, to extract software process dependencies and mode-specific behaviors through static source code analysis. Moreover, the source code may not always be fully accessible due to intellectual property constraints.

Nevertheless, most industrial dCPS include runtime monitoring and logging mechanisms to support debugging, diagnostics, and testing. This existing infrastructure can be used to collect execution traces containing detailed runtime data collected from real system executions. Traces record events such as process activations, message exchanges, and synchronization points, providing a practical and scalable source of

system behavior insights across different system operation modes. Moreover, such system execution traces could be leveraged to derive abstract system models in an automated way. This enables scalable model derivation across heterogeneous subsystems requiring insignificant manual engineering effort and very limited (or no) access to the application software source code. Such trace-based model derivation approach is particularly valuable for complex industrial dCPS, where static analysis of the source code is infeasible.

However, the usefulness of system execution traces for automated system model derivation depends on the type of collected data, its level of detail, and whether it reflects the system behavior across relevant scenarios (system operation modes). The right level of detail and the type of runtime data to be collected are crucial factors in making the trace-based model derivation both accurate and scalable. This leads to the third challenge:

CHL3: Identify the runtime system execution data to collect, and specify how to use it for automated derivation of the highly abstract and sufficiently accurate application software and hardware platform models needed for DSE of complex industrial dCPS.

After obtaining the abstract models, we need to represent them in an appropriate form which allows performance analysis. There are generally two approaches to achieving this: translating the models into analytical representations for analytical-based performance analysis or transforming them into executable representations for simulation-based performance analysis.

Analytical approaches, such as using queueing models or formal verification techniques, operate on mathematical abstractions to derive performance metrics. These techniques are effective and sufficiently accurate when the system structure and behavior are static and predictable to a large extent, thereby allowing strong simplifying assumptions. However, such conditions rarely hold in large-scale industrial dCPS, where software components are developed independently, run on heterogeneous platforms, and exhibit mode-dependent behavior that varies across operational configurations. These systems often rely on event-driven communication and synchronization, and involve tight coordination among hundreds of processes. Analytical approaches struggle to accommodate this level of behavioral variability and fine-grained timing complexity, making them inadequate for evaluating realistic industrial scenarios. Attempting to

translate such complex systems into overly simplified analytical models risks losing essential details, leading to inaccurate performance insights.

In contrast, simulation approaches do not rely on static and strict structural assumptions. Instead, they evaluate the system through the replay or generation of representative execution scenarios, including varying workloads in different operation modes. This flexibility allows simulation approaches to faithfully capture event-driven interactions, runtime variability, and concurrent execution behavior. Additionally, in many cases, simulations scale effectively with system complexity. They support the exploration of a wide range of design alternatives by reusing the same abstract models defined per operation mode, while varying some parameters in the mapping and/or hardware platform models. This is particularly valuable in DSE, where the objective is not to evaluate a single design instance, but to assess a large set of possible architectures and configurations. For these reasons, a simulation approach is adopted in this work as the foundation for performance analysis in DSE. To enable simulation-based performance analysis, the abstract models must be transformed into executable simulation entities, thereby introducing the next challenge:

CHL4: Generate executable entities from the abstract models that are suitable for rapid simulation which enables efficient performance analysis of complex industrial dCPS.

Finally, while existing DSE studies for dCPS primarily focus on analyzing and optimizing the hardware platform and the assignment of software processes to hardware processing components, they often overlook a crucial factor: the internal structure and execution order of the software workload itself. The execution order of software processes and tasks within every process significantly impacts dCPS performance, particularly in terms of exploiting potential parallelism. Specifically, two types of parallelism are essential:

1. **Inter-process parallelism** which involves parallel execution among different software processes;
2. **Intra-process parallelism** referring to parallel execution of computation and communication tasks within a single process.

To make these types of parallelism analyzable and amenable for exploration during DSE, the application software model must capture both inter- and intra-process dependencies at a suitable level of abstraction. Such a model not only enables DSE

to identify and explore opportunities for parallel execution, but also ensures that any increase in hardware parallelism, such as adding more hardware processing components in the cyber part of dCPS, can be effectively utilized by exposing sufficient degree of software-level parallelism. This is particularly valuable for large-scale software, running on industrial dCPS, that is often proprietary and expensive to significantly modify but may still benefit from performance improvements through better scheduling and hardware utilization. This leads to our final challenge:

CHL5: Automatically derive an abstract application software model that captures both inter- and intra-process dependencies in order to enable the analysis, exploration, and exploitation of available parallelism without significantly modifying the original software infrastructure in complex industrial dCPS.

1.4 Research Contributions

This thesis addresses the five scientific challenges presented in Section 1.3 by devising, developing, and validating a comprehensive methodology, called **CompDSE**, for scalable and efficient DSE tailored to the computing (cyber) parts of complex industrial dCPS. The methodology integrates system-level modeling, automated model derivation and transformations, and simulation-based performance analysis into a coherent and tool-supported workflow.

To address challenge **CHL1**, the thesis introduces CompDSE as a structured DSE methodology composed of three main components: abstract models, an integrated toolbox, and simulation-based environment. The abstract models are derived from runtime execution traces, collected from an existing industrial dCPS, where DSE is applied to explore and design the next-generation alternatives of the dCPS. The toolbox includes algorithms for automated model derivation from execution traces, model-to-model transformations to convert the derived model into executable entities, and simulation code generation. These components enable the performance evaluation of each design configuration through discrete-event simulation.

In response to challenge **CHL2**, we devise three core models: an *application workload model* representing the software infrastructure of dCPS, a *delay-based hardware platform model* representing the hardware infrastructure of dCPS, and a *workload-to-platform mapping model* specifying the distribution of the software workload on the hardware

platform. These models are defined at a very high level of abstraction while maintaining sufficient accuracy for system-level performance evaluation of complex industrial dCPS.

The abstract application workload model captures the software workload as a set of processes that exchange messages via communication channels. For each dCPS operation mode, a process is represented as a time-agnostic sequence of computation and communication events that reflects the software execution flow without relying on platform-specific information. Every event has an abstract signature that models the workload of the event imposed on the hardware platform. Besides the workload signature, communication events include the size of the message which is communicated. Since we can derive the model from several execution traces representing different dCPS operation modes, the model captures mode variability in dCPS, enabling different mode-related software workloads to be modeled under varying configurations.

The abstract delay-based hardware platform model captures the hardware platform topology and, when transformed to executable entities, emulates computation and communication delays associated with the corresponding events in the abstract workload model. The hardware platform model consists of *hosts* connected to one or more *switches*, where each host contains multiple computing cores characterized by specific operating frequencies. While hosts are responsible for emulating the delays of computation events, switches emulate the delays of communication events as network transfer delays associated with exchanged messages between processes over a communication network.

The workload-to-platform mapping model specifies the assignment of processes in the application workload model to the hosts in the hardware platform model, thereby relating these two core models to form an abstract dCPS system model. In addition to the three core models, we have developed and incorporated in CompDSE an *environmental influence model* to account for timing effects originating from subsystems that could not be traced due to system constraints. These subsystems, which typically interact with the physical environment (e.g., sensors, actuators, or mechanical components), are not explicitly modeled. However, their interaction with traced components (hosts) introduces timing variations that influence the overall system performance. Therefore, the environmental influence model complements the abstract dCPS system model by implicitly and abstractly modeling the untraceable subsystems.

To address challenge **CHL3**, we study thoroughly traceable components within complex industrial dCPS and observe that the runtime data needed to derive the aforementioned abstract models is spread across both the user space and kernel space of the operating system (OS) running on these components. Therefore, the OS is

selected and utilized as the source for obtaining execution traces that combine standard kernel-space events with custom user-space events generated via software source code instrumentation.

In the user space, traces are collected by adding trace points in key library functions such as send and receive, that are commonly used by software processes and located in shared libraries. This approach avoids the need to instrument the source code of individual processes. The collected user-space traces include event timestamps, process names, and details of the instrumented functions, such as the size of the messages transferred by processes. These traces are primarily used to derive the processes, communication channels, and computation and communication events in the abstract application workload model. They also contribute to the derivation of the mapping model and parts of the abstract hardware platform model.

In the kernel space, relevant standard kernel events are collected to derive the abstract workload signatures of the events in the workload model and to provide complementary information for deriving the abstract hardware platform model. Having execution traces with the collected data, described above, we devise and implement several algorithms to automatically derive the three core abstract models utilized within the CompDSE methodology. These algorithms are included in the CompDSE toolbox mentioned earlier.

In addressing challenge **CHL4**, we develop and utilize model-to-model transformation algorithms, included in the CompDSE toolbox, that transform the automatically derived abstract application workload, platform, and mapping models into executable simulation entities. These executable entities include process entities for the application workload model, a centralized mapper entity representing the mapping model, and host and switch entities for the hardware platform model. The process entities generate and send computation and communication events to the mapper entity, which dispatches them to the appropriate host entities according to the specified mapping. Communication events are routed from the source host through the switch entities to the destination host. All these entities can interact and synchronize with the help of any discrete-event simulation engine, where host and switch entities simply emulate computation and communication timing delays only in order to support efficient and scalable system performance evaluation. In the CompDSE toolbox, we provide code generation algorithms that convert these executable entities into a specific code format targeting the well-known OMNet++ discrete-event simulation environment [113].

Finally, in response to challenge **CHL5**, we propose an important refinement of the abstract application workload model in order to capture both inter- and intra-

Chapters vs. Challenges	CHL1	CHL2	CHL3	CHL4	CHL5
Chapter 3: CompDSE Method+Models	✓	✓		✓	
Chapter 4: CompDSE Tools			✓	✓	
Chapter 5: CompDSE Evaluation				✓	
Chapter 6: Workload Model Refinement					✓

Table 1.1: Relation between the thesis chapters and research challenges

process parallelism. The most essential feature of the refined model is the grouping of cohesive computation and communication events into semantically meaningful tasks within every process that potentially could run in parallel. Each task corresponds to a sequence of events that need to be performed in response to an internal or external trigger that fires a process. Moreover, we propose a dependency analysis approach to identify opportunities for safely reordering and executing these tasks in parallel, without requiring significant modifications to the source code of the software infrastructure in dCPS. This approach enables deeper insights into the concurrency characteristics of the software workload and reveals optimization opportunities for better hardware resource utilization.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 provides the necessary background to understand the thesis contributions. It introduces the principles of DSE, the Y-chart modeling approach, and the limitations of current DSE tools when applied to complex industrial dCPS. It also presents the ASML TWINSKAN machine as a representative industrial dCPS and discusses the nature of event-driven execution in such systems. Additionally, it explains the distinction between user-space and kernel-space execution traces and outlines general strategies for collecting them, providing foundational context for the trace-based model derivation presented in later chapters. It also introduces some existing facilities used to support modeling, transformation, and simulation activities throughout the thesis.

Chapters 3 through 6 present, in great detail, the main research contributions that directly address the scientific challenges introduced in Section 1.3. Table 1.1 lists these chapters and shows the research challenges addressed in each chapter.

Chapter 3 describes the proposed CompDSE methodology, defines its abstract models, and explains how these models are represented as executable entities suitable

for discrete-event simulation, thereby addressing challenges **CHL1** and **CHL2**, and partly challenge **CHL4**.

Chapter 4 presents the toolbox of CompDSE. More specifically, it explains in detail the specific user-space and kernel-space execution traces and describes the algorithms for automatically deriving the abstract models from these execution traces, thereby addressing challenge **CHL3**. In addition, it introduces the algorithms to transform the derived abstract models into executable simulation entities and to generate simulation-ready code targeting OMNet++, thereby addressing challenge **CHL4**.

Chapter 5 evaluates the accuracy and efficiency of the CompDSE methodology by applying it to the ASML TWINSCAN machine. The efficiency is measured by recording the time required to evaluate individual design configurations. Additionally, through a set of manual DSE experiments, the chapter demonstrates the methodology's capability to evaluate the performance of alternative hardware platform designs. This confirms that the CompDSE methodology is sufficiently fast and practical for DSE in industrial dCPS, thereby addressing the efficiency aspect of challenge **CHL4**.

Chapter 6 presents the refinement of the abstract workload model to capture both inter- and intra-process dependencies. It also describes in detail the dependency analysis approach to identify opportunities for parallel execution of tasks within processes, thereby addressing challenge **CHL5**.

Finally, Chapter 7 concludes the thesis by summarizing the main contributions and outlining directions for future work.

