IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.22994

A Method for Building Trustworthy Hybrid Performance Models for Cyber-Physical Systems of Systems

MAHTAB MODABER[1], MARTIJN HENDRIKS[2], MARC GEILEN[3], TWAN BASTEN[4], JEROEN VOETEN[5]

[1]Electrical Engineering Department, Eindhoven University of Technology, The Netherlands
[2]Electrical Engineering Department, Eindhoven University of Technology, The Netherlands
[3]Electrical Engineering Department, Eindhoven University of Technology, The Netherlands
[4]Electrical Engineering Department, Eindhoven University of Technology, The Netherlands
[5]Electrical Engineering Department, Eindhoven University of Technology, The Netherlands

Corresponding author: Mahtab Modaber (m.modaber@tue.nl).

This work was supported by the TRANSACT EU project, https://transact-ecsel.eu/. TRANSACT has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Belgium, Denmark, Finland, Germany, Poland, Netherlands, Norway, and Spain.

ABSTRACT The increasing dynamism and interconnectivity of Cyber-Physical Systems of Systems (CPSoS) emphasize the need for advanced performance modeling approaches. Performance models serve the purpose of exploring and evaluating design solutions, providing insights into system behavior during the early design phases, and managing performance risks during development. Despite the advancements in hybrid modeling within the field of performance modeling, the challenge of constructing models effectively and efficiently while ensuring their trustworthiness has not received much attention. This paper proposes a method, that is, step-by-step guidelines on using formalisms, techniques, and tools, to construct trustworthy hybrid models for analyzing and predicting stochastic timing performance of CPSoS. The method integrates design patterns for improved model design and development, alongside Bayesian calibration and statistical validation techniques to further enhance trustworthiness of the models. Although many individual techniques, formalisms and tools are available, our contribution lies in proposing a systematic method for achieving trustworthiness within the context of hybrid performance modeling and simulation. We concretize the method for cloud-based cyber-physical systems. Through two case studies, we show the efficacy of the proposed method for accurately predicting end-to-end latency of offloading imaging tasks to the cloud. The first case study showcases the application of design patterns in a practical scenario involving a cloud-based healthcare system—a collaboration with an industry partner in healthcare systems. The second case study has been implemented as a prototype to show the use of calibration and validation techniques with operational data in a laboratory context using an image-reconstruction application.

INDEX TERMS Bayesian calibration, cloud-based cyber-physical systems, design patterns, hybrid modeling, model validation, trustworthy performance models.

I. INTRODUCTION

A. MOTIVATION

Performance models can provide valuable insights into the dynamic nature of Cyber-Physical Systems of Systems (CPSoS). A System of Systems (SoS) refers to a network of independent and autonomous systems working together to provide relevant services. Cyber-Physical Systems (CPS) interact directly with their physical environment through sensors and actuators. Their integration into a Cyber-Physical System of Systems (CPSoS) introduces novel services surpassing individual CPS capabilities, but introduces dynamism and complexity for performance analysis [1]. CPSoS can be found in various domains, like transportation, healthcare, manufacturing, defense, and smart cities. A system such as an X-ray medical scanner, as shown in Fig. 1, provides a concrete example of a cyber-physical system. The current generation of such systems integrates data acquisition at the device tier. But plans are in place to deploy image processing

capabilities in the cloud, turning such systems into cloud-based CPSoS. Companies are incorporating cloud resources into their systems to save costs and increase flexibility. Varying network latency and data processing time, workload variation, and dynamic resource allocation contribute to the dynamism and variability of such cloud-based systems. These aspects significantly impact system performance metrics, including fluctuations and variability in response time, end-to-end latency, throughput, and resource utilization. Another intricacy of such systems is the interdependent nature of parallel tasks utilizing shared cloud resources. As multiple tasks run concurrently, their latency is inherently impacted by resource contention and allocation within the cloud environment. Therefore, conducting a comprehensive analysis of system performance before initiating development becomes imperative. Performance models provide insight into how an entire system responds to different inputs and stimuli. This, in turn, enables practitioners to predict performance, identify potential bottlenecks, optimize efficiency, and manage performance risks.

The state of the art in modeling and simulation has advanced significantly in recent years, with hybrid models emerging as a forefront approach [2], [3]. Hybrid models combine the strengths of different modeling techniques like agent-based, discrete-event, and system-dynamics models, allowing for a more comprehensive and accurate representation of systems compared to monolithic models [4]. Progress in hybrid modeling has particularly focused on *formalisms*, *techniques*, and *tools*. These three aspects provide key elements of what constitutes a modeling methodology to [5], [6]. ''Formalisms'' are means to embody models of a system, and their desired properties and to associate them with precise semantics. When we use formalisms to construct models, we call them modeling languages. If a formalism is specifically designed to describe what a system should do, we call it a (property) specification language. ''Techniques'' are mathematically founded ways to transform or analyse models (e.g., for model refinement, simulation, verification, and performance evaluation of models). ''Tools'' are user-friendly computer support to enable effective and efficient application of formalisms and techniques. A key research gap in the domain of hybrid models centers around the fourth key element of such a methodology, namely a *method*. ''Methods'' are step-by-step processes for the effective and efficient construction of trustworthy models. Methods assist the designer with developing feasible designs by providing a number of design steps that have to be completed in a certain order. Each design step is supported by guidelines, which capture experience with applying certain formalisms and techniques. Determining which formalisms and techniques are suitable for a design process, therefore, forms an essential starting point for developing a design method [5], [6]. To address this gap, we propose *a systematic method that combines design patterns, calibration and validation techniques to construct trustworthy performance models efficiently and effectively*. Design patterns offer well-established, structured solutions to common modeling challenges, providing re-usability, consistency, clarity, and flexibility. This significantly improves modeling efficiency by saving time and reducing the effort required for design and implementation. Furthermore, the inclusion of calibration and validation techniques enhances the trustworthiness of the model ensuring that both the modeling process and the resulting models are effective in producing reliable and accurate results.

FIGURE 1. A medical scanner [7]. The image processing for such a scanner may be deferred to the cloud.

B. CONTRIBUTION

Our contribution is a proposed step-by-step method to use formalisms, techniques and tools to construct trustworthy hybrid performance models for CPSoS. This method combines existing academic work with our practical experience in building performance models through various applied-research projects with high-tech industry.

Our method comprises the following key steps supported by highlighted techniques:

1) Semi-formal problem description.
2) Model-architecture creation.
3) Selection of modeling style (per model component).
   Techniques:
   - Hybrid modeling (Agent-based, Discrete-event, System dynamics, Stochastic).
   - Design patterns (Y-chart, Model observability, Progress abstraction, Load abstraction).
4) Model refinement.
5) Model verification.
   Techniques:
   - Discrete-event simulation
   - Model checking
6) Model calibration.
   Techniques:
   - Approximate Bayesian computation
   - Discrete-event simulation
7) Model validation.
   Techniques:
   - Statistical tests
   - Discrete-event simulation

A detailed explanation of the steps, along with the corresponding techniques used, can be found in Sec. III. The method is not strictly linear. There is flexibility between steps, allowing certain steps (e.g., 3 and 4) to be performed concurrently as needed. If the desired result is not achieved in Steps 5 and 7, modelers have the option to return to earlier steps for design and model modification.

As far as we know, literature on the methodical details of building trustworthy hybrid performance models does not exist [4]. Model trustworthiness heavily depends on the model's purpose and context, which may be hard to generalize. That said, we believe that in a restricted application area and problem scope, certain guidelines can be defined for constructing a trustworthy model using the building blocks and techniques identified in the state of the art. The objective of this study's use-case models is to predict the stochastic timing performance of a cloud-based CPSoS in the form of the median of the end-to-end latency distribution. We use end-to-end latency analysis for cloud-based CPSoS as a case study to concretize our method.

C. RELATED WORK

In this section, we review previous research in three distinct areas: hybrid modeling and validation, design patterns, and model calibration.

Hybrid modeling and simulation involve the integration of various modeling formalisms and techniques, including agent-based, discrete-event, and system-dynamics modeling and analysis. These formalisms and techniques are often employed to model different aspects of a system across different levels of abstraction. Brailsford *et al.* [3] and Mustafee *et al.* [4] conducted comprehensive reviews on the definition, purpose, research gaps, and future opportunities in the field of hybrid modeling and simulation. However, as noted by Eldabi *et al.* [8], many hybrid modeling attempts can be characterized as ad hoc and pragmatic without a clear and standardized method. Furthermore, Brailsford *et al.* [3] reveal that the validation process is not commonly reported for hybrid models. In an analysis conducted by Kar *et al.* [9], it becomes evident that a significant proportion of the papers falls short in terms of model validation. Whereas some publications do make efforts to validate their models, employing statistical tests, face validity [3], or a combination of both, there is a noticeable limitation in these studies due to the lack of comprehensive explanations regarding their chosen validation processes. Overall, as noted by [10], there is no universal algorithm that can be used to determine which validation technique should be used to validate simulation models. The choice depends on the aim of the models, specifically, the type of analysis that is intended. Another factor is that validation is situation-dependent, meaning that the choice of technique depends on the specific context in which the simulation model is being used. Our proposed method addresses these methodological gaps.

The concept of design patterns was expressed early in the work of Alexander on the engineering of buildings and planning of towns [11], and is extensively applied in, for instance, object-oriented design and programming [12] and agent-based simulation [13]. Few studies, however, focus on patterns for performance analysis of complex systems by discrete-event simulation (DES). The ones that we are aware of are the following. The work in [14], [15] shows how existing design patterns from [12] can be applied to solve common problems in DES. Also, in [16], [17], [18], existing object-oriented design patterns are applied to build models in the discrete-event specification (DEVS) formalism [19]. The Y-chart pattern for decomposition of a system in an application that is mapped to a platform is introduced in [20], [21] in the context of hardware/software co-design and quantitative analysis of embedded systems, respectively. In [22], a detailed design pattern, a refinement of the Y-chart, is presented for modeling interaction on shared resources of software-intensive embedded systems. In [23], two design patterns are presented for track-based flow-management systems. Finally, some design patterns for the SimEvents tool are presented in [24]. Our method incorporates several of the mentioned design patterns, including the Y-chart pattern.

Model calibration is the process of fine-tuning model parameters to ensure that predictions of the model align with observed data. Pimentel *et al.*, in [25], proposed a framework that allows for the calibration of abstract performance models using concrete measurements. Additionally, Parappurath *et al.* [26] provided formal definitions of the errors involved in the calibration and prediction process, along with their quantitative relationships. However, their focus does not extend to stochastic performance models. When dealing with stochastic models, calibration may entail comparing the statistical distribution of the system's response to (samples of) the simulation model's stochastic output distribution using statistical tests or statistical disparity measures [27]. This paper includes the calibration of stochastic hybrid models, with a particular emphasis on predicting timing performance. To address the challenge of estimating the values of model parameters in stochastic models, we employ a Bayesian approach in which parameters are represented by probability distributions. The complexity arises from the fact that the likelihood evaluation of hybrid models is analytically intractable. Several methods, including Approximate Bayesian Computation (ABC), as initially introduced by Beaumont *et al.* [28], and likelihood-free Markov chain Monte Carlo (MCMC) techniques [29], can provide viable solutions. Approximate Bayesian Computation (ABC), in particular, has gained prominence as a popular and effective approach for approximating the model parameters' distributions, especially when dealing with models exhibiting intractable likelihoods. Substantial developments and discussions on the ABC algorithm and techniques are available in the handbook of Sisson *et al.* [30]. We adopt ABC in our method.

D. OUTLINE

In Sec. II, we present two use-cases, one industrial case from the medical domain to motivate the proposed method and an

accessible laboratory setup to demonstrate the effectiveness of the method using operational data. Sec. III recalls criteria for model trustworthiness, introduces the overall method, and proposes concrete supporting techniques and tools suitable in the context of performance modeling of CPSoS. Three (classes of) techniques, design patterns, model calibration, and validation, are discussed in detail in Sec. IV, V, and VI, respectively. The scope of the method and challenges during the different method steps are described in Sec. VII. Finally, Sec. VIII concludes.

II. TWO EXAMPLES OF CLOUD-BASED CPSOS

A. INDUSTRIAL HEALTHCARE CASE

An X-ray scanner (see Fig. 1) uses a moving C-shaped arm with an X-ray sensor to take images under various angles of a patient lying on a table. Such a set of images can then be used to reconstruct a 3D image, which is used during the medical procedure. This software-based reconstruction is compute-intensive, and currently is deployed on local, dedicated hardware. These systems have timing constraints on the end-to-end latency for the image processing: for instance, the time between the end of image acquisition and the output of the 3D reconstruction should exceed a certain bound at most 0.1% of the time. This reconstruction function is mission critical: its failure (including untimeliness) does not directly harm the patient, but severely hampers the procedure.

Deploying software-based 3D reconstruction to the cloud has several advantages such as opportunities for new business models and easier management of software versions. An important challenge to be overcome, however, is the fact that the cloud is a separate system not under the control of the manufacturer of the X-ray system or the hospital operating the X-ray system. The timing of cloud workloads is inherently difficult to predict and the cloud may not always be available (due to connectivity problems or SLA restrictions). Therefore, it is important to have a good understanding of the performance aspects of migrating mission-critical services from the scanner to the cloud. Our study focuses on end-to-end latency of tasks offloaded to the cloud.

Fig. 2 illustrates the components of the platform layer of a cloud-based X-ray system. In this layer, the 3D reconstruction functionality is deployed on cloud computing resources. 'Cloud cast' is a component that is responsible for collecting images from scanners and transmitting them to the cloud, where the 3D reconstruction process takes place. Once the reconstruction is complete, the results are sent back from the cloud to be displayed through the reporting user interface.

B. LABORATORY CASE

In our research lab, we employ a system similar to the industry case from the healthcare domain, serving as a prototype for the acquisition of empirical data to facilitate experimenting with calibration and validation techniques. Our setup uses the open-source software package 'opensfm' [31], deployed as a container on our local cloud infrastructure, for the purpose of generating a 3D image from a set of 2D images. We

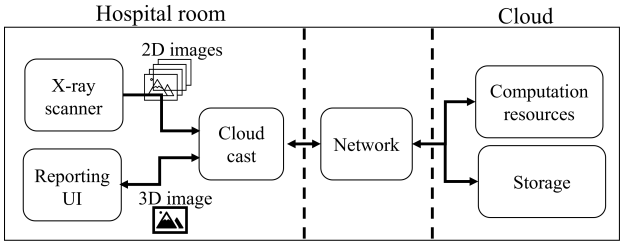

FIGURE 2. High-level system components in the industrial healthcare system

use 'Locust', a load testing tool [32], to simulate concurrent users generating reconstruction tasks. These simulated users transmit 2D images as input via the network to the 'opensfm' application, which subsequently processes the data and returns 3D images. We employ 'Zipkin' [33] for timing-performance data collection and 'Prometheus' [34] for usage data collection of resources (CPU and memory). See Fig. 3 for an overview of the setup.

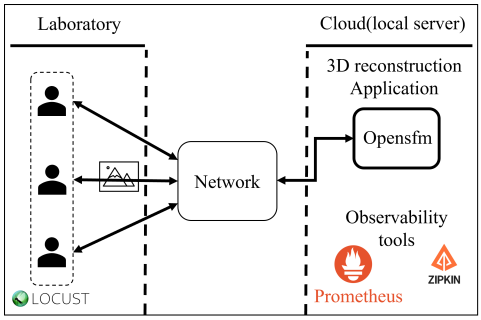

FIGURE 3. High-level system components in the laboratory setup

III. METHOD

This section first presents criteria for a trustworthy model. We then proceed with the proposed method to build such a model, providing formalisms, techniques and tools that support the method and that are applicable for performance analysis of CPSoS.

A. WHAT IS A TRUSTWORTHY MODEL?

We use the qualities that have been described in [35], [36]:

A. There is a clear object of modeling.
B. The model has a clear purpose.
C. The model is traceable.
D. The model is as truthful as needed.
E. The model is as simple as possible.
F. The model is extensible and reusable.
G. The model is inter-operable.

Given these criteria, we propose the following method.

B. A METHOD FOR BUILDING TRUSTWORTHY PERFORMANCE MODELS

The proposed method consists of seven steps, following the outline given in the introduction to this paper. For each step,

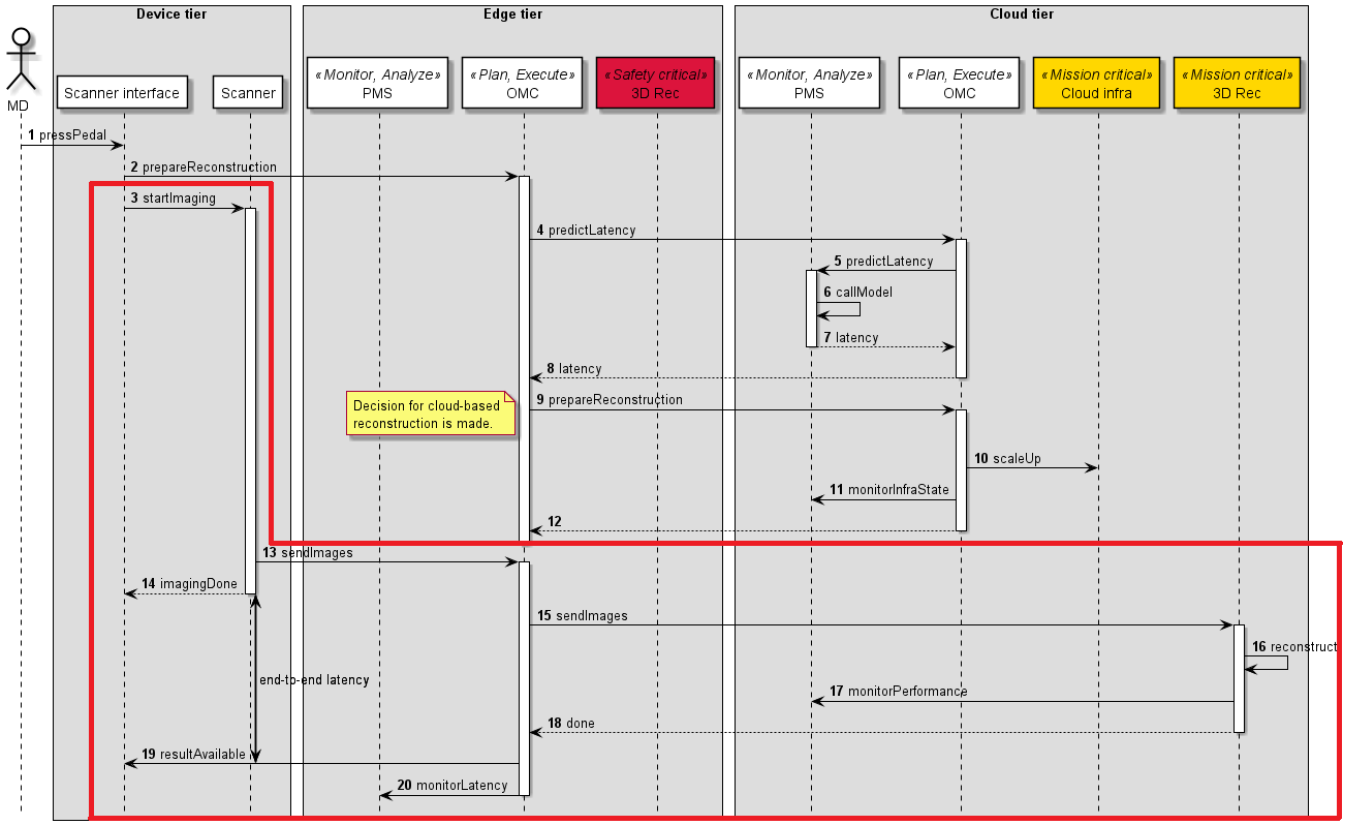

FIGURE 4. A high-level sequence diagram illustrating the message exchanges between components in the industrial healthcare case, created with PlantUML [37]

we not only explain the techniques, tools, and formalism, as applicable, but also offer detailed examples of their application to specific use-cases. Steps 1 through 5 are explained in detail in this section. Steps 6 and 7 are detailed in Sec. V and Sec. VI, respectively. These explanations should provide a clear understanding of how the method is practically applied within real-world scenarios.

Step 1: Semi-formal problem description

Description:

The first step in addressing a performance challenge is typically to create a semi-formal problem description in collaboration with domain experts. This description focuses on the performance-related issue at hand, offering a specific perspective on the system. This clarifies the object (A) and the purpose (B) of the model. Furthermore, this is the bridge between existing knowledge (in documents, other models, people's heads) and the formal performance model to be developed. As such, it is the starting point for traceability (C), the formal model and its output should be traceable to this semi-formal description, and from there to the existing knowledge. The A3AO (A3 Architecture Overview) [38] can be used, e.g., to create an architectural overview of the system performance aspect. It results in a concise overview (it fits on an A3-size sheet of paper) and has proven to be a good means to communicate with the different stakeholders. Languages such as UML [39] and SysML [40] can be employed during this stage to capture various system aspects such as structure and dynamics. [41] and [42] list the popular UML and SysML tools, respectively.

Example:

In this paper, the object of modeling is a cloud-based CPSoS. This includes the cloud infrastructure, and the applications that are running on it. It also includes the (many) clients of the system. We distinguish two high-level purposes for models that predict system performance. First, these models are used for risk reduction during system development, i.e., offline use. Second, these models are used online for performance management. Refinement of the purpose comes through specification of what exactly is to be predicted by the model. Fig. 4 shows a sequence diagram of the healthcare use-case that encapsulates a holistic flow of interactions among core components, encompassing the decision-making process required at the edge tier for executing the 3D reconstruction task on the cloud. Our industrial partner is primarily interested in the end-to-end latency of 3D reconstruction requests being offloaded to the cloud. The focus of the model being created therefore is on the highlighted segment of the system depicted in Fig. 4. This segment begins with the capturing of images by the scanner, involves their transmission to the cloud, and includes the reception of the resultant output. The model's

objective is to predict the end-to-end latency of executing the 3D reconstruction task on the cloud. For the industry use-case, we do not have operational data. So we cannot meaningfully concretize the purpose and required accuracy further at this stage. In the context of the laboratory case, where a similar high-level system architecture is employed, we choose the same modeling purpose, namely to predict the end-to-end latency of executing the 3D reconstruction. We require that the model's prediction accuracy is such that the median latency of at least 90% of the 3D reconstruction tasks align closely with the observed data. Additionally, the target is to achieve an accuracy of less than 1.0 second Mean Absolute Error (MAE) for the predicted median in task latency predictions. The median latency value in a system represents the typical time taken for data to travel from one end of the system to the other, indicating the central tendency of delay experienced within the system. Deviations from this median can highlight potential anomalies or issues, aiding in the identification of problems such as network congestion or hardware/software faults that may affect system stability. Understanding this metric facilitates the optimization of system components, software algorithms, and informed decision-making for resource allocation, system design improvements, and the selection of appropriate cloud services based on specific latency requirements. Typical aspects that need to be included for system timing performance are control and data flow, data sizes, computational loads, hardware speeds, buffer sizes, load scenarios, etcetera.

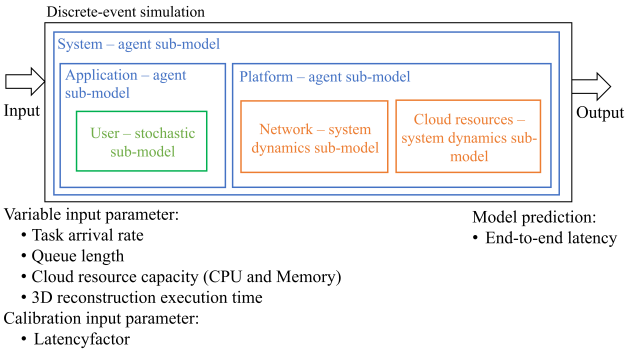

FIGURE 5. Model architecture and modeling style selection for the healthcare case.

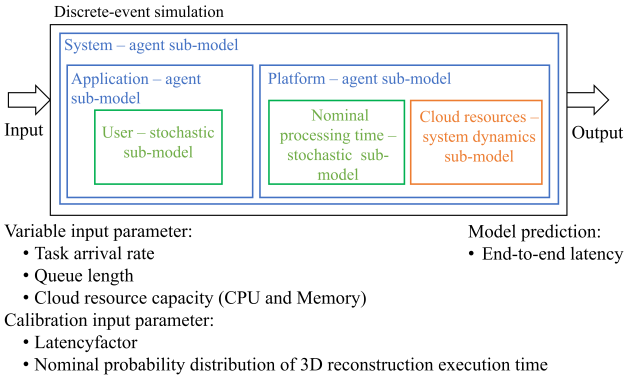

FIGURE 6. Model architecture and modeling style selection for the laboratory case.

Step 2: Model-architecture creation

Description:

The model components and their relationships are defined using the knowledge gained in Step 1. These components can be abstracted at different levels. In addition, model input parameters and model output need to be concisely specified. It's important to note that each model can have two categories of input parameters, variable input parameters (having known values) and calibration input parameters (having unknown values) [27]. The latter are used to calibrate the model against empirical data. The choices made in this step impact the simplicity (E), and the extensibility and reusability (F) of the model.

Example:

We have classified the model components into application and platform components, as depicted in Fig. 5 and 6, respectively. Within the application, we have identified the user component that models the creation of 3D reconstruction tasks. On the healthcare case's platform side (Fig. 5), the network component captures communication between the application and the cloud resources component. The cloud resources component models computational tasks. Model inputs are the reconstruction task arrival distribution, queue length, 3D reconstruction execution time, and the capacity of resources (CPU and memory). These are variable input parameters. Parameter 'Latencyfactor' is a calibration input parameter used for calculating delays. These choices for the model components and parameters are motivated by the specific requirements of the healthcare use-case. In the laboratory case, as depicted in Fig. 6, where the network delay in comparison to the 3D reconstruction processing time is negligible, we simplified the platform component by abstracting from details of the network and combining cloud resources and network into a single component, named the cloud resources. This component captures both communication and computation, including resource contention because of concurrent tasks. We incorporate a nominal processing time component to emulate variations in nominal 3D reconstruction execution time per task. This provides additional calibration parameters compared to the healthcare case.

Step 3: Selection of modeling style (per model component)

Description:

Hybrid modeling encompasses various abstraction levels and different facets of systems at the same level, enhancing the model's effectiveness and accuracy in fulfilling its purpose. The concept of hybrid modeling has evolved considerably from its initial focus on combining discrete and continuous methods. Its modern definition encompasses the utilization of two or more techniques from different modeling and simulation categories to create something novel. This novel entity

amalgamates the defining attributes of these techniques, enhancing the utility of the underlying modeling and simulation effort [4], [43].

In the following, we explain four of these techniques and their applications. Agent-based modeling is a technique used for modeling systems that consist of autonomous agents interacting with each other. These agents exhibit behaviors governed by simple rules and engage in interactions with other agents, influencing their behaviors in return [44]. Agent-based modeling is especially suitable for modeling infrastructures, because it is capable of representing the technical components as they are organized in a network, as well as the social components that can also be organized in a network. The interplay between the social and technical components leads to dynamic behavior and a dynamic system structure. Agent-based modeling finds applications in various domains and disciplines such as smart manufacturing [45], autonomous driving [46], surveillance [47], and critical infrastructure [48]. In surveillance, for instance, drones, targets and environments can be identified as agents, while the way drones navigate these environments and monitor them using sensors and information sharing can be identified as behaviors. Similarly, in smart manufacturing, products, machines, and workers can be considered agents, with inspection, scheduled operation on products, and manual tasks identified as behaviors.

In discrete-event modeling, the operation of a system is represented as a chronological sequence of events. Events change the state of the system, which often has a fixed structure, and its entities. These state changes trigger new events. Discrete-event modeling is mainly used to analyze the operation of systems, such as the flow of products in a manufacturing setup, the handling of data in an IT system, the handling of containers at ports, or the operation of trains in a railway network [49]. Key to discrete-event modeling is the representation of entities and their processing flow through the system, where they stay in queues, are delayed, processed, claim and release resources, split, combine, etc.

System dynamics refers to a collection of mathematical modeling techniques to capture the non-linear evolution of system behavior over time. It borrows concepts such as flows, feedback loops and multi-loop structures from control theory. System dynamics modeling has applications across various domains. It is used to model the dynamics of stock levels and order rates in supply chain management [50] , analyze the impact of changes in production processes in manufacturing [51], understand the dynamics of traffic flow in [52], and predict and mitigate flood risks by analyzing water levels and flow rates in critical infrastructure [53].

Stochastic processes are means to quantify the dynamic relationships between sequences of random events. The modeling essentially needs to capture the random variables and uncertainty parameters playing a role in the system at hand. It brings the probability factor in the calculation, which determines every possible outcome. In a stochastic model, a range of potential outcomes is predicted, each weighted by its probability of occurrence. However, the phenomena themselves are not inherently stochastic or deterministic; the choice to model them as such depends on the observer's intentions and objectives [54]. Analyzing the stochastic behavior of data traffic to optimize network performance and reliability in telecommunications [55], and understanding the impact of random climatic variations on ecosystems in environmental engineering [56], are some examples of the applications of this modeling technique.

The choice and combination among agent-based, discrete-event, system-dynamics, stochastic, and other modeling techniques can be determined by considering factors like the level of decision-making, available data, and the model architecture defined in Step 2. Design patterns, such as the Y-chart, model observability, progress abstraction, and load abstraction, enable implementation of architecture components with a hybrid modeling style. Design patterns are typical solutions to common problems [12]. They are discussed in detail in Sec. IV. These patterns can already be factored in during the semi-formal description of the system and may support the choice of modeling styles in this step. During the model refining step, they are used to create the formal system model. Step 3 impacts the truthfulness (D), extensibility and reusability (F) and interoperability (G) of the model.

Example:

Fig. 5 illustrates hybridization for the healthcare use-case by integrating an agent-based system-level model with system-dynamics and stochastic sub-models. Stochastic models are the preferred choice for capturing the variable 3D reconstruction request rates in the application agent. We employ system-dynamics models to capture the behavior of both the network latency and the execution time of the cloud-based computation. The models use the progress-abstraction design pattern discussed later. The models provide a high-level, abstract representation of the network and cloud platform that is particularly useful for large-scale design-space exploration during early design phases. Since the laboratory use-case is intended to experiment with and illustrate calibration and validation techniques, we assume that each 3D reconstruction task has a variable nominal execution time when processed in the cloud. That is, we decide to include a stochastic sub-model within the platform agent, see Fig. 6. A system-dynamics sub-model following the progress-abstraction design pattern (see Sec. IV) ensures a proper load-based scaling of the nominal execution time.

Step 4: Model refinement

Description:

In this step, we formalize the behavior of the model components established in Step 2 following the modeling style chosen in Step 3. Unclarities in the informal description are resolved in this step. Moreover, abstractions are typically applied to balance detail, effort and truthfulness. This step impacts truthfulness (D) and simplicity (E) of the model, and also extensibility and reusability (F).

During the model refinement step, we are tasked with selecting suitable formalisms and tools for each sub-model. There is a choice between using multiple formalisms and tools or opting for a single formalism and tool that supports various modeling styles. One possible approach is co-simulation. Co-simulation involves using multiple simulators or tools, each specialized in modeling specific aspects of the system, working together. This approach allows for a comprehensive representation of the CPSoS by leveraging the strengths of different tools or simulators. While employing multiple formalisms and tools can enhance the flexibility of representing diverse system components, it might complicate the subsequent analysis due to integration challenges. Conversely, choosing a single comprehensive tool could streamline the analysis process. For instance, a tool like AnyLogic [57], known for its versatility, supports many modeling styles within an integrated simulation framework. A potential risk of choosing a single tool is that it might limit flexibility in capturing specific system aspects. Ultimately, the decision on whether to employ multiple or a single comprehensive set of formalisms and tools depends on the trade-offs between modeling complexity, analysis needs, system comprehensiveness, and the specific requirements of the CPSoS being modeled.

Example:

Given the importance of stochastic aspects in our CPSoS use-cases, a formalism like POOSL (The Parallel Object Oriented Specification Language) [58] is a suitable choice for the modeling formalism for the use-cases. The complexity, scale, and inherent uncertainty of components in these systems necessitate an expressive formalism capable of supporting stochastic modeling and efficient analysis. POOSL models specify discrete-time Markov chains, which are suitable for capturing various stochastic effects. POOSL supports various modeling styles, through a language for specifying concurrent processes communicating through synchronous message passing, the process layer, and object-oriented data structures, the data layer. A probabilistic structural operational semantics based on Markov decision processes is defined for the process layer and a probabilistic denotational semantics for the data layer [59]. The availability of formal performance analysis techniques [5], grounded in its precise semantics, makes POOSL well-suited for modeling stochastic performance aspects, including estimating the expected end-to-end latency of requests in cloud-based CPSoS. Moreover, POOSL embeds a high-performance Discrete Event Simulation (DES) engine capable of handling industrial-sized problems, reinforcing its suitability for complex stochastic modeling scenarios in these systems. Fig. 5 and 6 illustrate our choice for POOSL/DES as a comprehensive modeling and analysis tool through the surrounding discrete-event simulation boxes.

Step 5: Model verification

Description:

We use the definition of model verification that is given in [60], [61]: ''Verification is the process of determining that a model implementation accurately represents the developer's conceptual description of the model and its solution''. A commonly used technique to verify whether a model adheres to the developer's needs is simulation. Techniques such as code review and documentation ensure consistency between a system's implementation and the conceptual model, providing further insight to the developer. Unit testing and integration testing are employed to verify the functionality of individual modules and their interactions within the hybrid model, ensuring their seamless cooperation. Moreover, model simulation and re-producibility tests verify the consistency and reliability of a model's outputs under various scenarios. Model checking [62] can be used to systematically examine whether a model satisfies formal specifications or properties. These verification techniques collectively contribute to assessing the reliability and adherence of models to specified requirements, including those outlined in Steps 1 to 3, aiding in agreements and decisions related to the conceptual model. This step can be based on combination of manual and automatic inspection of the model and its output, and contributes to traceability (C) and truthfulness (D) of the model.

Example:

As mentioned, the POOSL tool provides a simulation engine that can be used for model verification. POOSL can be complemented with the TRACE tool from the Eclipse TRACE4CPS project [63]. TRACE can be used to visualize simulation traces as Gantt charts (including events and continuous real-valued signals). Visualization plays a key role in the Model Observability pattern explained in Sec. IV; it supports traceability and the model verification step. Using Gantt charts and leveraging TRACE features, we may assess critical properties within the model. We can examine the duration of 3D reconstruction tasks, monitor queue status, and analyze the timeline of events. These analyses aim to ensure alignment with the expected functionality of the model.

Step 6: Model calibration

Description:

Calibration is defined as the ''process of adjusting numerical or physical modeling parameters in the computational model for the purpose of improving agreement with experimental data.'' [61]. This process depends on data captured from a prototype, and contributes to truthfulness (D). The model calibration step benefits from the use of Bayesian inference techniques. The 'PyMC' probabilistic programming library for Python [64] is a robust tool to facilitate this process. This step is further explained in Sec. V.

Step 7: Model validation

Description:

We use the definition of model validation that is given in [60], [61]: ''Validation is the process of determining the degree to which a model is an accurate representation of the real-world from the perspective of the intended uses of the model. The

goal of validation is to quantify confidence in the predictive capability of the model by comparison with experimental data''. Discrete-event simulation serves as a valuable technique for predicting timing performance metrics. Statistical testing techniques can be employed to compare operational data obtained from a prototype and simulation model output. The 'scipy.stats' Python library serves as an illustrative example tool for conducting statistical tests. This step contributes to truthfulness (D) of the model. It is further elaborated and explained for our laboratory use-case in Sec. VI.

Note that Steps 6 and 7 of the method require operational data, which is exclusively obtainable from a prototype. Steps 1-5 can already be used to create models in early phases of development. Although these might not be fully trustworthy models, they can still be useful in predicting general trends in the dynamic behavior and as such provide valuable input for the architecture and design of CPSoS.

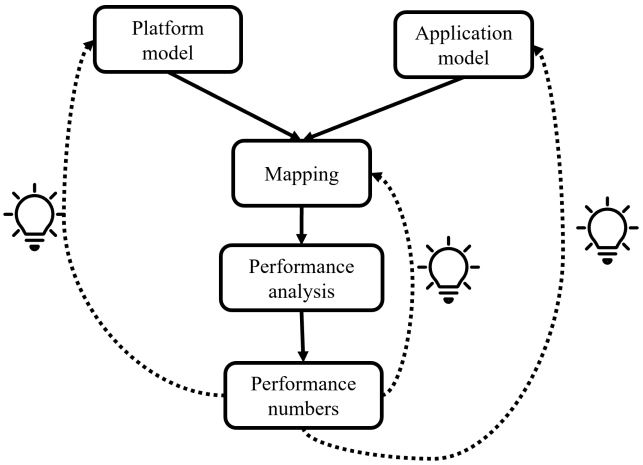

FIGURE 7. The Y-chart pattern: The designer analyzes performance numbers to propose improvements (light bulbs) of the platform, application, and/or mapping.

IV. PATTERNS FOR PERFORMANCE MODELS

A design pattern is a typical solution to a common problem and has four ingredients [12]. First, the *pattern name* provides us with a common vocabulary. Second, the *problem* describes the context and the problem at hand. Third, the *solution* explains an abstract, template-like approach to solving the problem. Fourth, the *consequences* list the results, advantages and drawbacks of using the pattern. We use this template to describe four patterns for hybrid performance models. In the *solution* part, we provide examples from literature to demonstrate the applicability of the pattern. Moreover, we conclude each pattern description with *examples*, both from the two use cases described in Sec. II and for other potential uses in the CPSoS domain.

A. PATTERN NAME: Y-CHART

Problem

A performance model should capture functional aspects of the system such as data dependencies between processing steps, but also the timing of those processing steps. Often, resources that determine the timing behavior are shared and can be used concurrently. Modeling the impact of resource sharing in a dynamic context is far from trivial.

Solution

The Y-chart pattern decomposes a system into an application, a platform, and a mapping of the application to the platform [21]. This separates the modeling of functional aspects (in the application) from the modeling of the timing behavior (in the platform). In Fig. 7, the application model describes the structure and characteristics of the application, including its tasks, resource requirements, and dependencies. The platform model defines the resources (e.g., CPU, memory, communication network), their properties, and algorithms for managing them (e.g., scheduling and allocation). The mapping model provides a global view of the load that tasks impose on the resources, which is needed to perform for instance scheduling. The mapping allows to translate task requirements and resource status into effective task execution performance. The Y-chart pattern for performance modeling and design space exploration is widely used across industries and areas [65], [66], [67]. In the embedded systems domain, the Y-chart design approach fits well with programmable architectures where the same resources can be reused for different applications through reprogramming [68].

Consequences

The Y-chart pattern makes a model flexible and modular, which makes it easier to maintain and extend over time (Req. (F) in Sec. III-A). This pattern is a precondition for the Progress Abstraction pattern explained below.

Examples

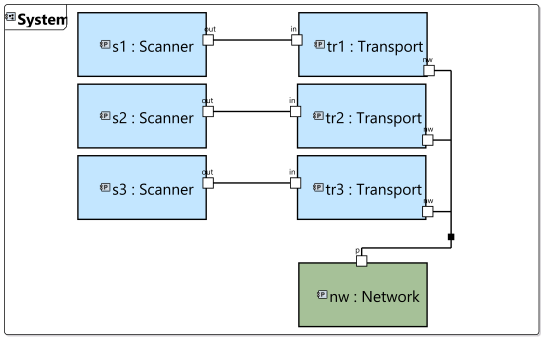

FIGURE 8. Y-chart pattern in the POOSL model.

Fig. 8 shows a structural representation of part of the POOSL model of the healthcare case. There are three scanners in a hospital that all use the same network to upload images to the cloud. The `Scanner` and `Transport` components are *application* components, and the `Network` is a *platform* component. The `Transport` components are *mapped* to the `Network`. Each `Transport` component has a certain *load*,

e.g., 2 MB of data, and the `Network` component takes care of processing this load by modeling the time that this takes.

Other potential use cases are, for instance, in smart manufacturing processes like assembly and quality control, where throughput and defect rates are of interest. The production processes (applications) are mapped to resources, including hardware, software, and network components, such as robotic arms, inspection cameras, IoT devices, and sensors. Similarly, in autonomous driving, applications like object and lane detection, path planning, and traffic rule compliance are mapped to resources such as cameras, control software, and networks. Y-chart-based performance analysis may then be used to analyze and optimize, for instance, response times.

B. PATTERN NAME: MODEL OBSERVABILITY

Problem

Modeling often requires many iterations to get to a model that behaves as intended ("model hacking precedes model checking" [69]). Model verification typically is done based on a manual inspection of the model in combination with inspection of its dynamic behavior. For both modeling and model verification, proper visualization and *tracking of the state of domain entities over time* is necessary. Traceability of the visualization to the concepts in the informal description is important for the domain experts who play a key role in the verification step. Out-of-the-box visualizations that are provided by tooling typically work on the concepts in the model, however. These do not necessarily have a one-to-one correspondence to concepts from the domain, e.g., the application and platform used in the Y-chart pattern.

Solution

Observability is the ability to understand the internal state or condition of a model by observing its external outputs. The better observable a model, the easier it is to identify root causes of performance issues without additional testing or coding. The dynamic behavior of the model can be monitored according to the three pillars of observability: (i) logs, (ii) traces, and (iii) metrics [70]. This often requires an external tool for the visualization aspects. The model is annotated such that it generates required input for the visualization tool when a simulation run is executed. Ref. [71] demonstrates the application of observability for monitoring distributed infrastructure, and [72] illustrates the use of this design pattern in software engineering.

Consequences

Proper observability leads to better traceability and easier modeling and model verification. Observability provides visibility for faster, automated problem identification and resolution. This pattern helps effective monitoring, troubleshooting and debugging by annotating models. A drawback is that the essence of the model is mixed with tracing information, which may reduce the readability of the model.

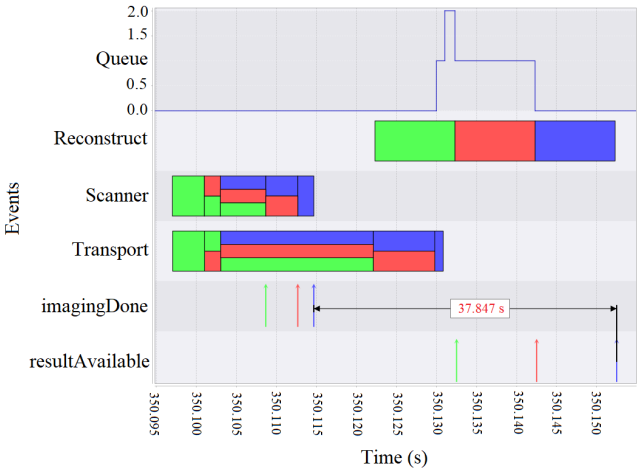

FIGURE 9. A detail of a POOSL simulation trace visualized by TRACE.

Examples

As explained in Sec. III, the POOSL tool can be extended with the TRACE tool for richer visualization (and analysis) of a simulation trace. The TRACE integration gives an API that can be used in the POOSL model to create customized visualizations. Fig. 9, for instance, shows a part of a POOSL simulation trace of the healthcare case as visualized by the TRACE tool. The Gantt chart shows a single *metric*: the size of the reconstruction queue over time in the swim lane at the top. Furthermore, it shows claims on resources (the colored rectangles) that represent the `Scanner`, `Transport` and `Reconstruction` tasks from example Sec. IV-A. It also shows two types of events (the colored arrows), which indicate the moments in time when the imaging is done and the reconstruction result is available, respectively. TRACE allows us to inspect the detailed dynamics, which helps modeling and model verification. Note that consistent naming over the various steps in the method highly contributes to traceability (C).

In other areas, such as surveillance applications using drones or driver assistance applications in the automotive domain, models capturing application functionality, computation and communication, and available resources can be annotated for monitoring and troubleshooting long response times.

C. PATTERN NAME: PROGRESS ABSTRACTION

Problem

Computation and communication in CPSoS can work on a fine level of granularity: processes that share a CPU or bandwidth of a communication link are finely interleaved. An example from the healthcare case is that an image stream of hundreds of images is needed for a single 3D reconstruction. Explicit modeling on this fine-grained level, however, often results in a model that does not scale.

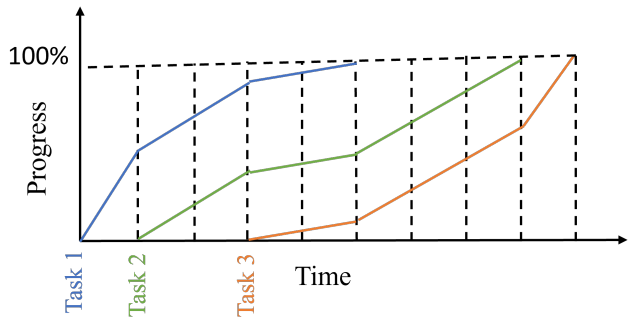

FIGURE 10. A linear progress abstraction for three tasks running partly simultaneously on the same resource.

Solution

This pattern assumes that the Y-chart pattern is used. The main concept behind the progress abstraction pattern is that a task receives a certain fraction of a resource, which determines its rate of progress. The fraction and the resulting task speed are determined by the platform model, which captures all resource scheduling rules, and which has access to the resource properties, including the resource performance. First, fine-grained application tasks (or fine-grained repetitions of a task) are condensed into a single, abstract application task. Second, a (piecewise-)linear progress abstraction (Fig. 10) is applied in the platform part of the model that abstracts the timing of the interleaving [22], [73], e.g., halving the speed of execution when two tasks share a processor.

Consequences

Application of this pattern results in a more scalable model. A potential drawback is the loss of accuracy because of the abstraction.

Examples

A single 3D reconstruction requires hundreds of images to be uploaded to the cloud via the `Transport` tasks in example Sec. IV-A. Modeling this explicitly does not scale, as the full model has thousands of scanners. Uploading a stream of, say, 300 images of 2 MB each, can be condensed into uploading a single blob of data of 600 MB. The `Network` resource model uses the piecewise-linear progress abstraction for the concurrent uploads. In case of three concurrent uploads, for instance, each one gets one third of the available bandwidth. Fig. 9 shows three overlapping `Scanner` tasks and also three overlapping `Transport` tasks from example Sec. IV-A. The timing of the `Transport` tasks is determined by the `Network` model.

In the automotive domain, a charging station for multiple electric vehicles may be effectively modeled using the progress abstraction pattern. Each vehicle receives a certain fraction of the total available power from the charging station, determining the rate at which each vehicle charges. Such a model may, for instance, be used for analyzing completion times of charging vehicles or, with a proper load model (as discussed next), for utilization ratios and waiting times of vehicles.

D. PATTERN NAME: LOAD ABSTRACTION

Problem

Cloud-based CPSoS architectures are star shaped, with the cloud in the middle, and with many clients around it. The internal behavior of the clients can be complicated and this may cause scalability issues for the analysis of the model.

Solution

Load abstraction refers to the simplified modeling of requests (including groups of clients or service requests) in the form of manageable and understandable representations. This abstraction allows modelers to analyze and predict the behavior of systems under various load conditions without dealing with every minute detail of the actual loads. The internal behavior of a group of clients is abstracted into a set of requests to the cloud, rather than including every detail of the clients' behavior. For example, the behavior of a subset of hospitals in our healthcare case can be abstracted into a sequence of timestamped 3D reconstruction requests to the cloud. Other types of abstractions, e.g., fitting a distribution of requests to the cloud based on sampling the behaviors of a hospital, are also possible. By using a load abstraction, we do not need to observe every single action; instead, we can use statistical techniques to approximate their collective behavior. Load abstraction finds its application for instance in order picking within manufacturing, where numerous individual requests and interactions generated by the picking process can be abstracted and simplified using the load abstraction pattern [74].

Consequences

For the creation of the abstraction we still need to analyse all internal behavior of the clients, which may take significant time. Once the abstraction has been created, however, it can be reused often in the same model and variants of the model.

Examples

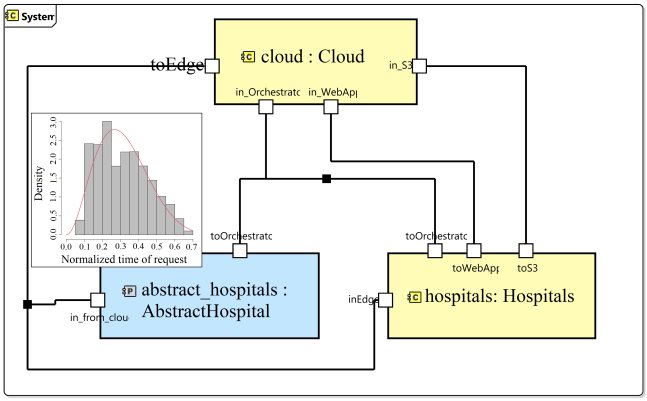

FIGURE 11. Abstraction of load of a subset of hospitals into a Beta distribution.

Fig. 11 shows part of the POOSL model for the healthcare case. The cloud component serves thousands of hospitals. A subset of hospitals is abstracted into the `AbstractHospital` component. A probability distribution specifies how the requests of the abstracted subset are distributed over time. The `Hospitals` component still contains detailed internal behavior of a single hospital to estimate the end-to-end performance of 3D reconstruction requests for that hospital.

Other examples of potential uses of the load abstraction pattern occur, for instance, in surveillance, where the occurrences of incidents may be abstracted in the form of distributions, or in the already mentioned example of a charging station, where the arrivals of vehicles over time may be abstracted through distributions.

V. MODEL CALIBRATION

Model calibration is a pivotal step in trustworthiness of computational modeling and simulation. Its primary objective is to minimize discrepancies between simulated outcomes and real-world observations or measured data obtained from prototypes. This process involves adjusting calibration model parameters to align the model's outputs with the actual behavior of the system. The calibration step directly impacts the truthfulness (D) of the model, one of the criteria outlined in Sec. III-A.

According to definitions in [27], [75], model input parameters generally fall into two categories: variable input parameters and calibration input parameters. Variable input parameters are assumed to have known values and can be varied during the calibration process. They describe different configurations or designs used by the simulation model to predict system performance. Calibration input parameters, on the other hand, have unknown values but remain constant across the various configurations or designs for which the model is calibrated. These parameters often represent physical aspects of the system being modeled, such as growth rates, probabilities, constants, coefficients, or thresholds. Sometimes, they serve as abstract representations of underlying processes within the system rather than having direct physical interpretations. Additionally, there might be tuning calibration parameters unique to the simulation model that regulate the model's behavior. The output of the simulation model depends on both variable and calibration input parameters. For example, the hydrological and water quality models of [76], have diffusion and pollutant degradation coefficients as calibration parameters. In the intelligent driver model of [77], two important calibration parameters are the minimum gap distance from the leading vehicle and the minimum time interval between the following vehicle and the leading vehicle. In [78], hyper-parameters such as the shape and magnitude of model inadequacies were used as calibration parameters.

The values attributed to calibration parameters can be unknown, or lack a clearly defined optimal value that minimizes differences between the model's output and the observed data. Combining prior knowledge of these parameters with available real data from a prototype can help to estimate their optimal values. This process involves using prior knowledge, which may come from expert judgment, historical data, or theoretical considerations, to inform initial estimates of the parameters. By integrating this prior information with empirical data collected from a prototype or real-world system, we can refine these estimates. The Bayesian approach, for instance, uses prior distributions to represent our initial beliefs about the parameters. As new data is collected, the Bayesian framework updates these beliefs to produce posterior distributions. These posterior distributions reflect our updated understanding of the parameter values, balancing prior knowledge with the evidence provided by the real data. This technique provides a systematic way to incorporate both theoretical insights and empirical observations, leading to more accurate and reliable parameter estimates [75]. Bayesian inference involves using a search algorithm that continuously, for each set of calibration parameter values, generates simulation data from the model. These outputs are then compared to measured data acquired from a prototype. The objective is to collect a set of calibration parameters values that produces simulation outputs that maximally agree with the measured data from a prototype. Additionally, it seeks to quantify the uncertainty associated with these parameter values through credible intervals or probability distributions. In this context, the calibration parameters are considered as random variables with prior probability distributions, representing initial knowledge or beliefs regarding their values. These prior distributions are then combined with observed data from the real system or prototype using Bayes' theorem. This integration yields posterior distributions, providing an updated and more informed probability distribution of these uncertain calibration parameters after considering the observed data. By obtaining the posterior distribution for a calibration parameter, we not only get a point estimate of the parameter we are interested in, but also a measure of uncertainty of that estimate, which helps in making informed decisions and understanding the reliability of our estimates.

The estimation of a vector Θ of calibration parameters (the collective representation of multiple calibration parameters in a vector) for simulation model M, achieved by comparing simulation output with measured prototype data O, can be expressed in the following equation [79]:

$$p\left(\Theta|O,M\right) \propto p\left(O|M\left(\Theta\right)\right) \cdot p\left(\Theta|M\right) \tag{1}$$

where,

- $p\left(\Theta|M\right)$ is the prior distribution of vector calibration parameter Θ.
- $p\left(O|M\left(\Theta\right)\right)$ is the likelihood function providing the probability of observing data O given the vector calibration parameter Θ in the model.
- $p\left(\Theta|O,M\right)$ is the posterior distribution of the vector calibration parameter Θ given observed data O and model M.

We have broken down the calibration process following (1) into four steps as follows:

1) Specifying prior distributions of calibration model parameters.
2) Determining likelihood function $p\left(O|M\left(\Theta\right)\right)$.
3) Approximating posterior distributions Θ.
4) Estimating values for calibration parameters.

Specifying prior probability distributions involves defining our initial beliefs or uncertainties about parameter values before observing data. These distributions represent the range of plausible values and uncertainty. They are based on expert or modeler knowledge, historical data, constraints or theoretical considerations. For instance, a uniform prior distribution assigns equal probability to all values within a specified range, implying that each value within that range is equally likely before considering any data. This least informative distribution is often used when there is no strong prior knowledge favoring specific parameter values, treating all values within the range as equally plausible. Uniform priors are particularly useful when there's a lack of information or when it is desirable to avoid biasing the analysis toward any particular value.

The likelihood function quantifies the probability of observing the given data under specific model parameters, effectively measuring how well the model explains the data. Determining the likelihood function is important step because it combines with the prior distribution to form the posterior distribution, updating our beliefs about the parameters in light of new evidence. When dealing with hybrid models, determining and evaluating the likelihood function can be computationally intractable or analytically prohibitive [80]. However, this does not have to be the case if we can generate synthetic data instead. The Approximate Bayesian Computation (ABC) technique becomes valuable in situations where we lack a precise likelihood expression [28], [29], [30]. The term 'approximate' in ABC refers to the lack of an explicit likelihood, but not to the use of numerical methods to approximate the posterior, such as Markov chain Monte Carlo or Variational Inference. Another commonly used term for the ABC technique is 'likelihood-free' technique. From the perspective of the ABC technique, the generator of synthetic data, or in our case the simulation model, is a black-box. We feed parameter values at one side and get simulated data from the other [81]. The basic notion of ABC technique is to replace the likelihood evaluation by a function δ that computes a distance metric or, more generally, some form of discrepancy between the observed data O and the synthetic data generated by a parameterized model simulation M [81]:

$$p\left(\Theta|O,M\right) \overset{\infty}{\sim} \delta\left(O,M\left(\Theta\right)|\epsilon\right) \cdot p\left(\Theta|M\right) \tag{2}$$

ϵ is a tolerance parameter because the chance of generating synthetic data being equal to the observed data is virtually null for most problems. Selecting the appropriate ϵ often involves a balance between the desired precision of the estimation and the computational resources available. It is common to start with a relatively large ϵ and progressively reduce it while monitoring the convergence and stability of the results before settling on a final value.

ABC employs various distance metrics, such as Euclidean distance, Manhattan distance, Mahalanobis distance, Minkowski distance, and others. The selection of a distance metric is contingent upon the specific application, aligning with the modeling objectives and the distinctive traits of the data being compared—such as dimensionality and sparsity [82], [83]. The decision regarding a distance metric revolves around choosing measures adept at addressing the challenges posed by high dimensionality, data sparsity, computational complexity, and the interrelationships among calibration parameters more effectively than alternative distance metrics.

In the third step of the Bayesian technique, where we aim to approximate posterior distributions of Θ, several inference techniques are available. Obtaining posterior distribution analytically is often challenging or infeasible, especially for hybrid models or high-dimensional parameter spaces. This is where sampling techniques come into play. These techniques range from grid-based approaches to sophisticated techniques like Metropolis-Hastings [84], Hamiltonian Monte Carlo, and Variational Inference. Sequential Monte Carlo (SMC) also known as particle filters [85], is a versatile and commonly employed choice for dynamic models within the context of the ABC technique [86], [87]. The SMC technique stands out for its adaptability to dynamic systems and efficient handling of complex, high-dimensional posterior distributions that might be multi-modal, irregularly shaped, or have discontinuities. Therefore, we also opt for the use of this technique to approximate posterior distributions in our use-case.

Once the posterior distribution of parameters has been approximated, concrete values for the parameters need to be selected for the simulation model in the fourth step of the calibration technique. Maximum A Posteriori (MAP) values represent the most probable values of the parameters given the observed data and the prior distributions. MAP is the parameter value for which the posterior distribution is maximal ($\arg\max_\theta$). These values are suitable for use as the calibrated values in the simulation model because they provide a point estimate of the parameters that are most likely to produce the observed data.

Example:

Identifying calibration parameters falls within the responsibility of the modeler. In the laboratory use-case, which serves as a prototype for the healthcare case study, we have identified four calibration parameters characterized by uncertain values in the vector model parameter $\Theta = [\theta_0, \theta_1, \theta_2, \theta_3]$ (see Table 1). The first parameter is the 'Latencyfactor', which we use in the calculation of the delays. The other three parameters collectively use to define the distribution of the nominal 3D reconstruction execution time. Our only prior knowledge is on the boundaries of the distribution. So we start with uniform distributions, as shown in Table 1. The data involved in the comparison of simulation output and observed data is high-dimensional due to the fact that every simulation produces over 70 distinct end-to-end latency values, each correspond-

TABLE 1. Four calibration parameters, priors and most probable value of posteriors

Parameter	Name	Prior	MAP
θ_0	Latencyfactor	$U(4,5)$	4.479
θ_1	Min	$U(91,95)$	94.086 (s)
θ_2	DeltaMode	$U(1,3)$	1.207 (s)
θ_3	DeltaMax	$U(1,3)$	1.298 (s)

ing to a unique 3D reconstruction task. Recognizing this characteristic of the data, as highlighted in research [83], we made the decision to employ the Manhattan distance metric. This metric, given in (3), effectively gauges the disparity between simulation output and observed prototype data in the second step of the calibration process.

$$\delta\left(O, M\left(\Theta\right)\right) = \sum_{i} \left|O_i - M\left(\Theta\right)_i\right| \tag{3}$$

where i uniquely identifies each 3D reconstruction task. $M\left(\Theta\right)_i$ represents the end-to-end latency values collected during simulation runs, providing a basis for comparison with observed prototype data O_i. Initially setting ϵ to 1, we iteratively reduced it to 0.1 during the ABC process, adjusting the acceptance threshold for simulated data to better match the observed non-normally distributed end-to-end latency patterns. Upon approximating the posterior distribution of calibration parameters, Fig. 13 showcases both the 2D joint posterior distributions and the 1D marginalized posterior distribution of these parameters. This visualization is particularly valuable for models with multiple calibration parameters, allowing exploration of correlations and joint distributions between these variables. In Bayesian statistics, HDI stands for the Highest Density Interval, which is a credible interval that contains a specified portion of the posterior probability distribution and has the property of having the highest density of probability within that interval. Fig. 13 represents the interval that contains 95% of the posterior probability density. This interval is chosen to encompass the highest density region of the posterior distribution based on the observed data and the prior information. From these distributions, we derive estimates for calibration parameters that exhibit the highest probability and offer the closest match to the observed data. The MAP (most probable) values for the calibration parameters are detailed in Table 1. These values are derived from joint posterior distributions that are considered most probable given the observed data. We use these values in the simulation model to generate data for the subsequent step of model validation.

VI. MODEL VALIDATION

Calibration minimizes the disparity between the output of the simulation model and the observed data from the prototype by fine-tuning calibration parameters. It is important to evaluate how accurately the model, with these adjusted and calibrated parameter values, predicts the system's performance and how reliable these predictions are. This step contributes to truthfulness (D) criteria outlined in Sec. III-A. Upon obtaining adjusted values for the model parameters through the calibration process, they are applied to the model to generate new output. This newly generated data, along with the measured data from the prototype, undergoes a validation process. During the validation process, the goal is to assess the degree of similarity and accuracy between the predictions generated by the calibrated model and the observed data obtained from the real-world prototype or system. If the validation results do not meet the desired criteria, which means that prediction of model is not close enough to real-world data, the process needs iteration for further refinement. This iterative procedure is illustrated in Fig. 12.

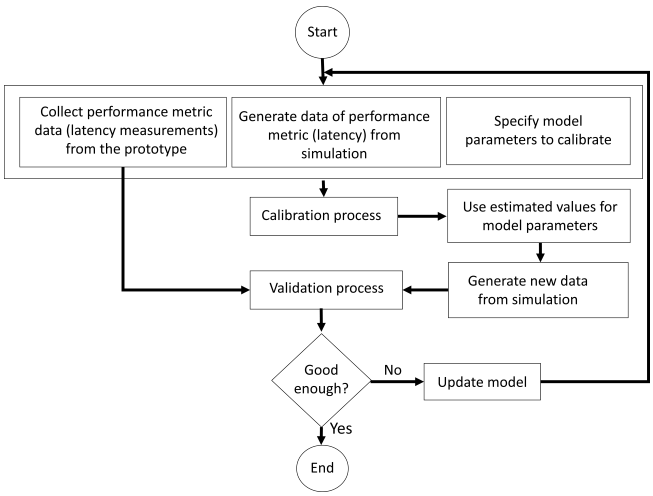

FIGURE 12. Calibration and validation procedure.

Due to inherent variability of complex systems like CPSoS, modeled with stochastic hybrid models, one cannot directly compare individual values between observed and simulated data, because one cannot expect an exact match between observations and model outputs. Instead, statistical tests serve as a valuable technique for validation in these scenarios. These tests focus on comparing not the specific values themselves but rather the statistical properties and distributions of the simulated data with those observed empirically. The goal is not to achieve an exact equivalence between individual observations but to evaluate if the statistical characteristics of the simulated data are sufficiently similar to the empirical data. These tests allow us to determine if the patterns, trends, and statistical properties—such as mean, variance, median, skewness, exhibited in the simulated data, adequately capture those observed in real-world empirical data [3].

Statistical tests broadly fall into two categories: non-parametric and parametric. Parametric tests assume the data to follow an approximate normal distribution, whereas non-parametric tests, also known as distribution-free tests, do not make assumptions about the data distribution. The choice of which test to employ depends on the analysis question concerning the model's intended application and the inherent characteristics of the data being analyzed. The parametric t-test is commonly used when specific assumptions like inde-

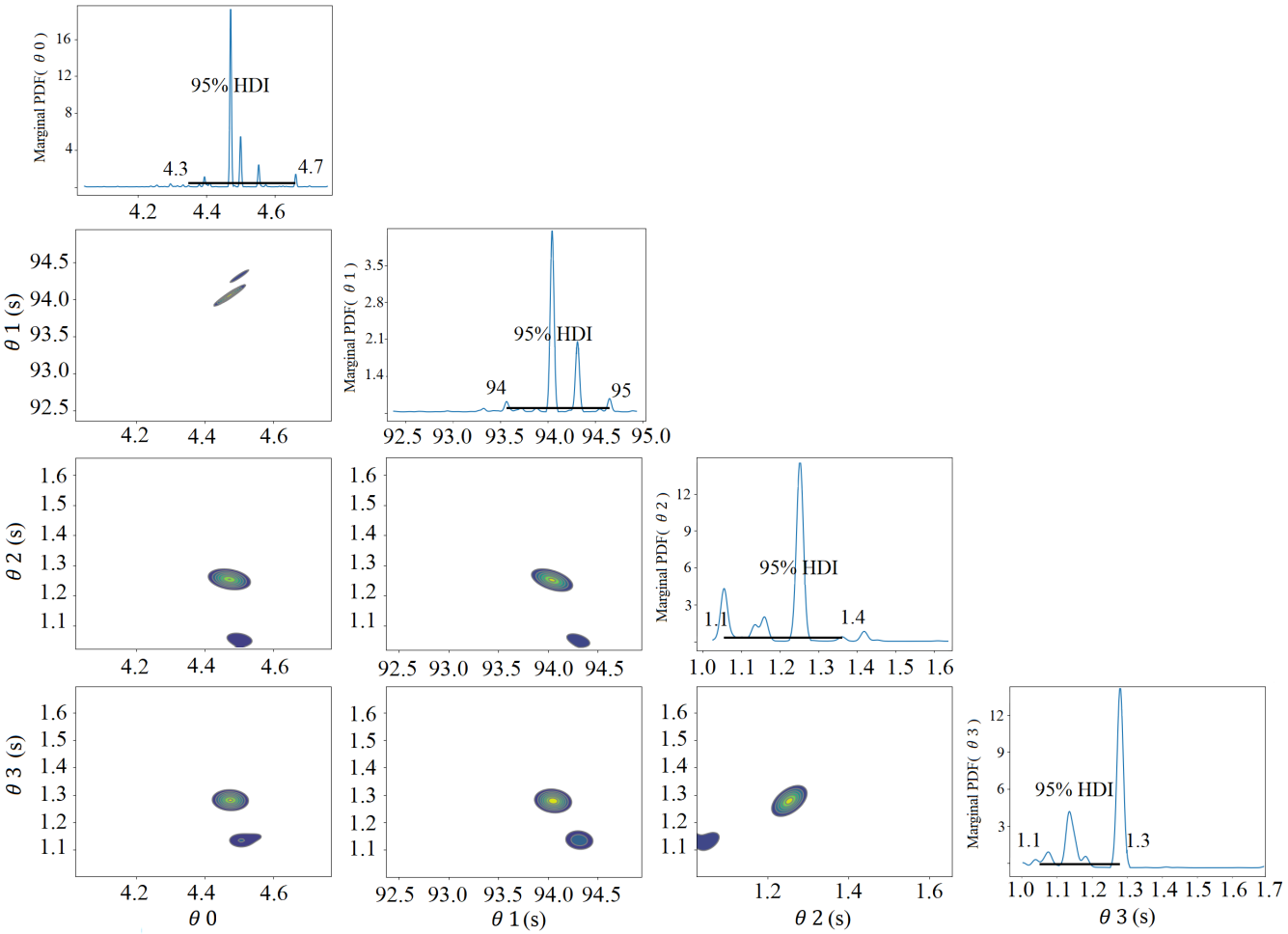

FIGURE 13. **This figure presents the joint posterior distribution and marginal distribution of calibration parameters, showcasing optimal values for each parameter indicated by lighter colors and highlighting their interdependence. Specifically, the joint distribution of θ_0 and θ_1 illustrates the relationship between these two parameters.**

pendence, homogeneity, and normality are met in two groups of samples. It's particularly suitable for analyzing questions related to the mean of simulated output distributions, assessing significant differences between the model output and observed data. In cases the data violate the assumption of normality, non-parametric tests like the Mann-Whitney U test (Wilcoxon rank-sum test) are often applied. This test doesn't assume a specific distribution within samples, relying on independence between compared groups [88]. When interpreting Mann-Whitney U test results concerning the medians of two data samples, assuming a similar shape of distribution is crucial. In the absence of assuming identical distributions between the groups, the test focuses on comparing the ranks of observations between the two groups, determining whether observations in one group significantly tend to have higher or lower ranks than the other group [89]. In the model validation process, distinguishing corresponding data sample groups from simulation model output and real data collected from a prototype is essential. This allows for a comparison of statistical properties such as mean, median, rank, and other relevant metrics. The goal is to identify differences in these properties that align with the model's research question. By focusing on identifying disparities between these statistical properties and assessing their relevance to the research question addressed by the model, it helps in evaluating how closely the model predictions agree to observed data. The accuracy of these statistical properties can be measured by metrics such as the Mean Absolute Error (MAE), aligning with research question and model purpose. This evaluation is pivotal in determining the reliability and trustworthiness of the model and outputs.

Example:

Data collection: The output generated by the simulation represents a series of end-to-end latencies linked to 3D reconstruction tasks, much like the data obtained through measurements in the prototype. Each 3D reconstruction task is treated as a random variable within this context. To capture the inherent variability within these reconstructions, the simulation model runs multiple times, producing a collection of samples

for each latency associated with the 3D reconstruction task. By aggregating these samples obtained from repeated simulation runs, we can describe and establish the distribution connected to each latency for the 3D reconstruction task. The same approach should be applied to the prototype. It necessitates repeating the experiment multiple times and gathering samples of the end-to-end latency corresponding to each 3D reconstruction task to derive the relevant distribution.

Data characteristics: The observations gathered from the prototype system and simulation model reveal that the end-to-end latency distribution for each 3D reconstruction task does not follow a normal distribution, as illustrated in Fig. 14.

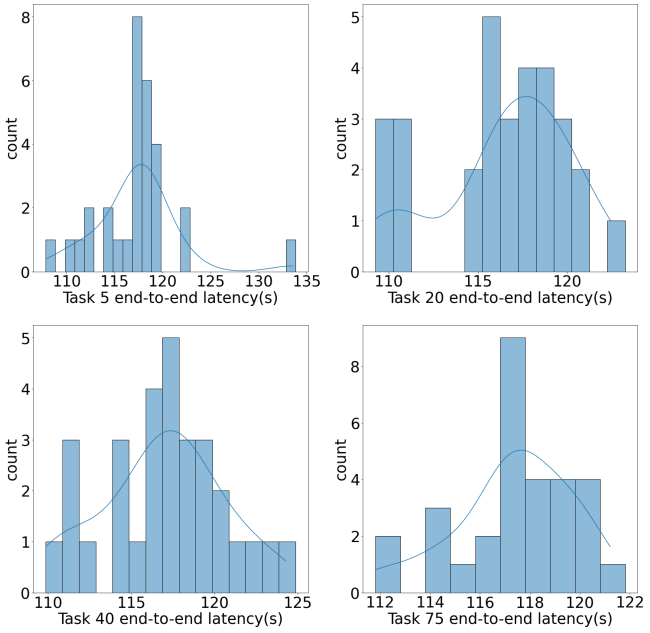

FIGURE 14. End-to-End latency histogram with distribution of task i=5, 20, 40, and 75 collected for n=30 experiments

Due to deviation from the normality assumption in both the measured data from the prototype and the output of the simulation model, we opted for non-parametric tests to compare the end-to-end latency distributions of individual 3D reconstruction tasks between the simulation model and measured prototype data. The Mann-Whitney U test is used to compare two independent distributions, particularly when the data do not adhere to normal distribution assumptions. When applying the Mann-Whitney U test to compare the distributions of end-to-end latencies associated with the random variables of 3D reconstruction tasks, we observed that in 93% of 3D reconstruction tasks (70 tasks out of 75), the median of the end-to-end latency distributions generated from the simulation model output was not significantly different from the corresponding measured output obtained from the prototype. Following our analysis, the model has achieved an MAE of 0.62 seconds for predicting the median end-to-end latency of these tasks. According to our model's objective for the laboratory case, which was specified in Step 1 in Sec. III-B, and based on the results of the validation tests, we confirm that the model predictions demonstrate no significant difference compared to the observed data from the prototype in more than 90% of the median latency distributions, and achieve an accuracy level of median latencies represented by an MAE (Mean Absolute Error) value of less than 1.0 second. Therefore, criterion (D) of model trustworthiness (The model is as truthful as needed), as defined in Sec. III-A, is quantified and considered fulfilled. Fig. 15 illustrates the median end-to-end latency distributions of some corresponding 3D reconstruction tasks, predicted by the model and measured from the prototype. The model predictions are tailored to their specific objectives, which is the median end-to-end latency in the task distributions. In conclusion, the model trustworthiness criteria are established following the steps in our method, and are confirmed through examples, showcasing the effectiveness and efficiency of the proposed method.

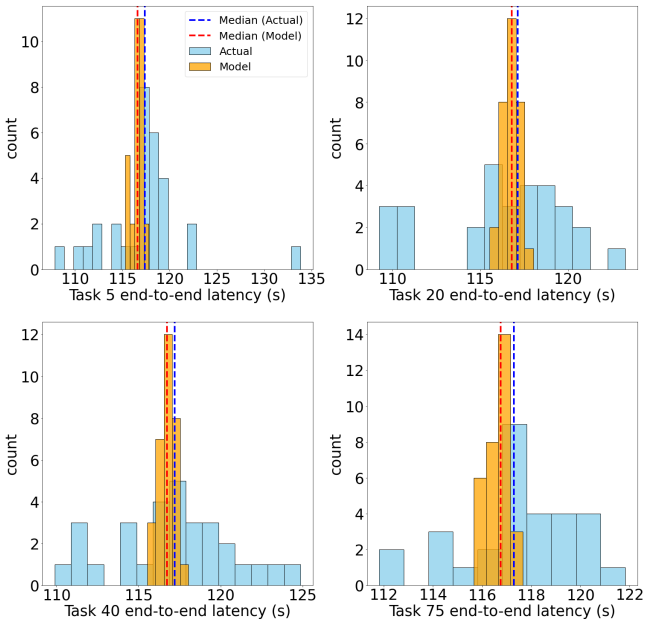

FIGURE 15. End-to-end latency histograms for 3D reconstruction tasks with indices 5, 20, 40, and 75 were collected from 30 experiments, for comparing model predictions and actual measurements from the prototype. The median end-to-end latency distributions between the model predictions and prototype measurements exhibit an accuracy level with a mean absolute error of less than 1.0 second.

VII. SCOPE AND CHALLENGES

In this section, we clarify the scope of our method, and potential challenges faced when applying it.

Our method is specifically designed to construct stochastic performance models to effectively handle uncertainty phenomena within the context of CPSoS, where dynamicity and variability are common. Its value may be less for systems without uncertainty, although its ingredients, for instance, the design patterns, may be meaningful to a wider class of systems.

Despite having a systematic method for support, the modeling process may still be challenging for designers with limited modeling experience. Challenges may arise in decomposing

system components, deciding about the appropriate level of abstraction for each component (which involves a trade-off between abstraction and model accuracy), and selecting suitable modeling styles and patterns for each. When applying the Y-chart design pattern, challenges include deciding how to map application components to resource components, for instance, ensuring scalability as the system grows. It is moreover important to define model variables that are in line with calibration and validation tests. For instance, in our lab use case, we require that the end-to-end latency random variables for the reconstruction tasks be independent from each other. It is essential to accurately define the distributions of these random variables because this affects decisions regarding valid statistical validation tests.

In the Bayesian calibration step, it is often challenging to determine the required complex likelihood functions, especially for hybrid models of CPSoS. Approximate Bayesian Computation (ABC) offers a flexible solution to this challenge, by avoiding the need for explicit likelihood functions. However, it may require many simulations and careful selection of tolerance. Despite scalability challenges, as the computational burden increases exponentially with the dimensionality of the problem, ABC remains valuable for Bayesian inference when traditional techniques are impractical. Additionally, efficient sampling techniques like Sequential Monte Carlo (SMC) help sample from posterior distributions, especially in high-dimensional spaces. However, the SMC technique can still be computationally intensive, particularly for large-scale problems. Scalability challenges arise from the varying number of calibration parameters across applications, requiring more time and resources for sampling. Careful parameter selection can mitigate this issue and improve efficiency in model behavior determination.

In the validation step, assumptions about the distribution of output data (performance metrics) determine the type of validation test that can be used among applicable statistical tests. It may be a challenge to find a suitable statistical test that fits with both the model assumptions and the model purpose. Parametric and non-parametric statistical tests differ in accuracy, leading to a trade-off between, possibly restricting, assumptions and the accuracy of the validation results. This trade-off impacts how closely the model predictions align with the output of the systems and needs careful consideration in the use of the method.

VIII. CONCLUSIONS

Our study introduces a systematic method tailored to construct trustworthy hybrid performance models for Cyber-Physical Systems of Systems (CPSoS) effectively and efficiently. A trustworthy model is defined by seven criteria. The proposed method is designed to fulfill these criteria. Constructing a trustworthy model for systems characterized by variability and uncertainty is a considerable challenge. The complexity arising from task parallelism and its subsequent impact on resource utilization and timing performance is addressed using proposed design techniques within our method. The incorporation of design patterns, such as the Y-chart, load and progress abstraction, and model observability, contributes to an efficient approach. Design patterns generally result in time and cost savings [12], [22]. Our method integrates hybridization, calibration, and validation techniques, effectively enhancing the modeling process and predictive capabilities. Determining the purpose of modeling before initiating the process is pivotal, as emphasized in the initial step of our method. This determination influences subsequent choices, particularly the selection of validation techniques. Moreover, defining the purpose establishes a basis for evaluating the model's trustworthiness.

We applied and evaluated our method in two case studies. In one case, the model underwent calibration and validation processes using operational data to ensure its reliability and accuracy in real-world scenarios. The primary objective in this use-case was to predict the median end-to-end latency distribution within these systems with a specific level of accuracy. By employing our method, we successfully predicted this timing performance metric close to empirical data, meeting our expectations.

The following are relevant future research directions:

- Alternative calibration techniques: Investigating and implementing alternative calibration techniques to improve the accuracy of the model's data generation. The comparison of these techniques will not only benchmark their efficiency but also assess their direct impact on enhancing the effectiveness of the model's output.
- Advanced validation techniques: Exploring additional validation techniques or integrating multiple validation techniques to quantitatively measure the trustworthiness of the model's predictions and outcomes. The development of a comprehensive assessment framework will provide a multi-perspective evaluation of the model's trustworthiness.
- Real-world application studies: Applying the method to diverse real-world case studies across various industries and scenarios. Such practical applications will support an assessment of the method's adaptability, validity, and performance in real-life contexts, offering valuable insights into its practical utility. We explained our method providing examples of potential uses in areas such as smart manufacturing, autonomous driving and transportation, and surveillance. It would be interesting to demonstrate the method's efficacy in addressing complex challenges related to dynamic behavior and uncertainty in these domains.
- AI techniques: Recent developments have shown the power of AI techniques. For our method, AI can potentially enhance sensitivity analysis by systematically and efficiently exploring the effects of varying input parameters on model outputs. This helps in identifying critical factors and their impact on the system being analysed. Additionally, AI can generate synthetic data to augment limited real-world data, improving the robustness of simulations. Finally, machine-learning-based models

may be a good addition to a hybrid model to capture aspects of components that are insufficiently understood or too complex for knowledge-driven modeling.

REFERENCES

[1] A. Bondavalli, S. Bouchenak and H. Kopetz, *Cyber-Physical Systems of Systems Foundations – A Conceptual Model and Some Derivations: The AMADEOS Legacy*, 1st ed. Springer, 2016.

[2] N. Mustafee, A. Harper and B.S. Onggo, "Hybrid Modelling and Simulation (M&S): Driving Innovation in the Theory and Practice of M&S," in *2020 Winter Simulation Conference (WSC)*, Orlando, FL, USA, 2020, pp. 3140–3151.

[3] S.C. Brailsford, T. Eldabi, M. Kunc, N. Mustafee and A.F. Osorio, "Hybrid Simulation Modelling in Operational Research: A State-of-the-Art Review," *Elsevier*, vol. 278, pp. 721–737, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221718308786

[4] N. Mustafee, S.C. Brailsford, A. Djanatliev, T. Eldabi, M. Kunc and A. Tolk, "Purpose and benefits of hybrid simulation: contributing to the convergence of its definition," in *Proc. of the 2017 Winter Simulation Conference (WSC)*, Las Vegas, NV, USA, 2017.

[5] B.D. Theelen, "Performance modelling for system-level design," Ph.D. dissertation, Dept. Elect. Eng., Eindhoven Univ., Eindhoven, The Netherlands, 2004.

[6] W.P.M.H. Heemels, G.J. Muller, *Boderc: model-based design of high-tech systems : a collaborative research project for multi-disciplinary design analysis of high-tech systems*. Eindhoven, The Netherlands: Embedded Systems Institute, 2006.

[7] Koninklijke Philips N.V, https://www.philips.com/a-w/about/news/media-library/20170516-Philips-Azurion-image-guided-therapy-platform-room-view.html.

[8] T. Eldabi et al., "Hybrid Simulation: Historical Lessons, Present Challenges and Futures," in *2016 Winter Simulation Conference (WSC)*. Washington, DC, USA: IEEE, 2016, pp. 1388–1403.

[9] E. Kar, T. Eldabi and M. Fakhimi, "Hybrid Simulation in Healthcare: A Review of the Literature," in *2022 Winter Simulation Conference (WSC)*, Singapore, 2022, pp. 1211–1222.

[10] R.G. Sargent and O. Balci, "History of verification and validation of simulation models," in *2017 Winter Simulation Conference (WSC)*, Las Vegas, NV, USA, 2017, pp. 292–307.

[11] C. Alexander et al., *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1995.

[13] F. Klugl and L. Karlsson, "Towards Pattern-Oriented Design of Agent-Based Simulation Models," in *Multiagent System Technologies*, P. P. L. Braubach, W. van der Hoek and A. Pokahr, Eds. Heidelberg, Berlin, Germany: Springer Berlin Heidelberg, 2009, pp. 41–53.

[14] A.V. Pristupa and O.A. Zmeyev, "Design patterns in discrete-event simulation (DES)," in *Proceedings. The 8th Russian-Korean International Symposium on Science and Technology, 2004. KORUS 2004.*, vol. 1, Tomsk, Russia, 2004, pp. 141–144.

[15] H. Bibin et al., "Application of design patterns in power system transient simulator," in *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, vol. 3, Zhangjiajie, China, 2012, pp. 465–469.

[16] A.E. Ferayorni and H.S. Sarjoughian, "Domain Driven Simulation Modeling for Software Design," in *Proc. of the 2007 Summer Computer Simulation Conference*. Society for Computer Simulation International, 01 2007, pp. 297–304.

[17] M.E. Hamri and L. Baati, "On Using Design Patterns for DEVS Modeling and Simulation Tools," in *Proc. of the 2010 Spring Simulation Multiconference*. Society for Computer Simulation International, 2010.

[18] M. Hamri, R. Messouci and C. Frydman, "Discrete Event Design Patterns," in *Proc. of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. New York, NY, USA: ACM, 2013, p. 349–354. [Online]. Available: https://doi.org/10.1145/2486092.2486138

[19] B.P. Zeichler, *Theory of Modelling and Simulation*. Wiley & Sons, 1976. [Online]. Available: https://api.semanticscholar.org/CorpusID:60034336

[20] F. Balarin et al., *Hardware-Software Co-design of Embedded Systems: The POLIS Approach*. USA: Kluwer Academic Publishers, 1997.

[21] B. Kienhuis, E. Deprettere, K. Vissers and P. Van Der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. IEEE International Conference on Application-Specific Systems, Architectures and Processors*, Zurich, Switzerland, 1997, pp. 338–349.

[22] M. Hendriks, T. Basten, J. Verriet, M. Brassé and L. Somers, "A Blueprint for System-Level Performance Modeling of Software-Intensive Embedded Systems," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 1, p. 21–40, Feb 2016. [Online]. Available: https://doi.org/10.1007/s10009-014-0340-3

[23] E.A. Lee and M. Sirjani, "What Good are Models?" *Formal Aspects of Component Software*, vol. 11222, pp. 3–31, 2018.

[24] MathWorks, "SimEvents Common Design Patterns," https://nl.mathworks.com/help/simevents/gs/common-design-patterns.html.

[25] A.D. Pimentel, M. Thompson, S. Polstra and C. Erbas, "Calibration of Abstract Performance Models for System-Level Design Space Exploration," *Journal of Signal Processing Systems*, vol. 50, no. 2, p. 99–114, 2008. [Online]. Available: https://doi.org/10.1007/s11265-007-0085-2

[26] V.V. Parappurath, J.P.M. Voeten and K.C. Kotterink, "Calibration Error Bound Estimation in Performance Modeling," in *2013 Euromicro Conference on Digital System Design*, Los Alamitos, CA, USA, 2013, pp. 97–102.

[27] A. Tolk, J. Fowler, G. Shao and E. Ycesan, *Advances in Modeling and Simulation: Seminal Research from 50 Years of Winter Simulation Conferences*, 1st ed. New York: Springer, 2017.

[28] M.A. Beaumont, W. Zhang and D.J. Balding, "Approximate Bayesian computation in population genetics," *Genetics*, vol. 162, no. 4, pp. 2025–2035, 12 2002. [Online]. Available: https://doi.org/10.1093/genetics/162.4.2025

[29] P. Marjoram, J. Molitor, V. Plagnol and S. Tavare, "Markov chain Monte Carlo without likelihoods," in *Proc. of the National Academy of Sciences*, vol. 100, no. 26, 2003, pp. 15 324–15 328.

[30] S.A. Sisson, Y. Fan and M.A. Beaumont, "Overview of Approximate Bayesian Computation," 2018.

[31] "OpenSfM," https://opensfm.org/.

[32] "Locust," https://locust.io/.

[33] OpenZipkin contributors, "Zipkin, a distributed tracing system." [Online]. Available: https://zipkin.io/, 2021.

[34] "Prometheus, an open-source monitoring and alerting toolkit." [Online]. Available: https://prometheus.io/.

[35] F. Vaandrager, "What is a Good Model?" [Online]. Available: https://www.cs.ru.nl/F.Vaandrager/PV/what-is-a-good-model.html, 2010.

[36] A.H. Mader, H. Wupper and M. Boon, "The Construction of Verification Models for Embedded Systems," *CTIT Technical Report Series*, vol. 1, no. TR-CTIT-07-02, 2007.

[37] A. Roques, "PlantUML, an open-source tool allowing users to create diagrams from a plain text language." [Online]. Available: https://plantuml.com/, 2009.

[38] P.D. Borches Juzgado, "A3 Architecture overviews: A tool for effective communication in product evolution," Ph.D. dissertation, Dept. Eng. ,Twente Univ., Twente, The Netherlands, Dec 2010.

[39] UML, *OMG Systems Modeling Language (OMG UML)*, Object Management Group Std., 2012.

[40] SysML, *OMG Systems Modeling Language (OMG SysML), Version 1.3*, Object Management Group Std., 2012.

[41] "SysML tools, Commercial, Free and Open Source SysML Modeling Tools," [Online]. Available: https://sysml.org/sysml-tools/.

[42] W.J. Thong and M. A. Ameedeen, "A Survey of UML Tools," in *Proc. of the International Conference on Data Engineering 2015 (DaEng-2015)*. Singapore: Springer Singapore, 2019, pp. 61–70.

[43] A. Tolk, E.H. Page and S. Mittal, "Hybrid simulation for cyber physical systems: state of the art and a literature review," in *Proc. of the Annual Simulation Symposium*. San Diego, CA, USA: Society for Computer Simulation International, 2018, pp. 1631–1645.

[44] C.M. Macal and M.J. North, "Tutorial on Agent-Based Modelling and Simulation," in *Proc. of the 37th Conference on Winter Simulation*. Orlando, Florida, USA: Winter Simulation Conference, 2005, p. 2–15.

[45] A. Fedorov, E. Goloschchapov, O. Ipatov, V. Potekhin, V. Shkodyrev and S. Zobnin, "Aspects of Smart Manufacturing Via Agent-based Approach," in *Procedia Engineering*, vol. 100, 2015, pp. 1572–1581, 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014.

[46] J. Li, E. Rombaut and L. Vanhaverbeke, "A systematic review of agent-based models for autonomous vehicles in urban mobility and logistics: Possibilities for integrated simulation models," *Computers, Environment and Urban Systems*, vol. 89, p. 101686, 2021.
[47] T.I. Zohdi, *The Game of Drones: rapid agent-based machine-learning models for multi-UAV path planning*. Springer, 2020, vol. 65, no. 1.
[48] J.R. Thompson, D. Frezza, B. Necioglu, M.L. Cohen, K. Hoffman and K. Rosfjord, *Interdependent Critical Infrastructure Model (ICIM): An agent-based model of power and water infrastructure*. The Netherlands: Elsevier, 2019, vol. 24.
[49] A.I. Potekhin, S.A. Branishtov and S.K. Kuznetsov, *Discrete-event models of a railway network*. USA: Plenum Press, Feb 2016, vol. 77, no. 2.
[50] B.K. Bala, F. Arshad, N. Mohamed and M. Kusairi, *Modelling of Supply Chain of Rice Milling Systems in Bangladesh*. Singapore: Springer Singapore, 2017.
[51] N. Kazantsev, O. Petrovskyi and J.M. Müller, "From supply chains towards manufacturing ecosystems: A system dynamics model," *Technological Forecasting and Social Change*, vol. 197, p. 122917, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162523006029
[52] P. Pfaffenbichler, "Modelling with Systems Dynamics as a Method to Bridge the Gap between Politics, Planning and Science? Lessons Learnt from the Development of the Land Use and Transport Model MARS," *Transport Reviews*, vol. 31, no. 2, pp. 267–289, 2011.
[53] T.D. Phan, E. Bertone and R.A. Stewart, "Critical review of system dynamics modelling applications for water resources planning and management," *Cleaner Environmental Systems*, vol. 2, p. 100031, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666789421000234
[54] H.M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*. https://doi.org/10.1016/C2009-1-61171-0: Elsevier, 2011.
[55] N. Gautam, *Stochastic models in telecommunications for optimal design, control and performance evaluation*, ser. Handbook of Statistics. Elsevier, 2003, vol. 21.
[56] G. Visconti, *Stochastic Weather and Climate Models*. Cham: Springer International Publishing, 2021.
[57] A. Borshchev, *Multi-method modelling: AnyLogic*. John Wiley & Sons, Ltd, 04 2014.
[58] P.H.A. van der Putten and J.P.M. Voeten, "Specification of reactive hardware/software systems : the method software/hardware engineering (SHE)," Ph.D. dissertation, Dept. Elect. Eng., Eindhoven Univ., Eindhoven, The Netherlands, 1997.
[59] L.J. van Bokhoven, "Constructive tool design for formal languages : from semantics to executing models," Ph.D. dissertation, Dept. Elect. Eng., Eindhoven Univ., Eindhoven, The Netherlands, 2002.
[60] AIAA, *Guide for the Verification and Validation of Computational Fluid Dynamics Simulations (AIAA G-077-1998(2002))*, 1998.
[61] B.H. Thacker, S.W. Doebling, F.M. Hemez, M.C. Anderson, M.C., J.E. Pepin, and E.A. Rodriguez, "Concepts of Model Verification and Validation," *OSTI.GOV*, 10 2004. [Online]. Available: https://www.osti.gov/biblio/835920
[62] C. Baier and J. Katoen, *Principles of Model Checking*. New york, USA: MIT Press, 01 2008, vol. 26202649.
[63] "Eclipse Trace4cps," https://www.eclipse.org/trace4cps.
[64] "PyMC," https://www.pymc.io.
[65] F. Kautz, H. Blume and C. Sauer, "Methodology for an Early Exploration of Embedded Systems using Portable Test and Stimulus Standard," in *2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, Porto Alegre, Brazil, 2022, pp. 1–6.
[66] M. Herget et al., "Design Space Exploration for Distributed Cyber-Physical Systems: State-of-the-art, Challenges, and Directions," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 2022, pp. 632–640.
[67] J. Lapalme, B.D. Theelen, N. Stoimenov, J.P.M. Voeten, L. Thiele and E.M. Aboulhamid, *Y-chart based system design: a discussion on approaches*. Montreal, Canada: Montreal Univ., 2009, pp. 23–56.
[68] B. Kienhuis, Ed.F. Deprettere, P. van der Wolf and K. Vissers, "A Methodology to Design Programmable Embedded Systems," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation — SAMOS*, J. T. Ed.F. Deprettere and S. Vassiliadis, Eds. Heidelberg, Berlin, Germany: Springer-Verlag, 2002, pp. 18–37.
[69] H. Brinksma and A.H. Mader, "On Verification Modelling of Embedded Systems," *CTIT technical report series*, no. 04-03, 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:16074298
[70] C. Sridharan, *Distributed systems observability: a guide to building robust systems*. O'Reilly Media, Inc, 2018.
[71] R. Gatev, *Observability: Logs, Metrics, and Traces*. Berkeley, CA, USA: Apress, 2021.
[72] T. Savas and K. Gokmen, "Relational Logging Design Pattern," *Computers, Materials & Continua*, vol. 75, no. 1, pp. 51–65, 2023.
[73] R. Jonk, J. Voeten, M. Geilen, T. Basten and R. Schiffelers, "Timing Prediction for Service-Based Applications Mapped on Linux-Based Multi-core Platforms," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Prague, Czech Republic, 2018, pp. 130–139.
[74] K. R. T. van Gils, A. Caris and K. Braekers, "Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse," *European Journal of Operational Research*, vol. 277, no. 3, pp. 814–830, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221719302516
[75] M.C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models (with discussion)," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00294
[76] B. Bai, F. Dong, W. Peng and X. Liu, "Bayesian-based calibration for water quality model parameters," *Water Environment Research*, vol. 95, no. 10, p. e10936, 2023.
[77] C. Zhang and L. Sun, "Bayesian Calibration of the Intelligent Driver Model," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2024.
[78] M. Kathrin, H. Yeonsook and C. Ruchi, "Influence of error terms in Bayesian calibration of energy system models," *Journal of Building Performance Simulation*, vol. 12, no. 1, pp. 82–96, 2019. [Online]. Available: https://doi.org/10.1080/19401493.2018.1475506
[79] J.S. Speagle, "A Conceptual Introduction to Markov Chain Monte Carlo Methods," 2020.
[80] N. Jagiella, D. Rickert, F.J. Theis and J. Hasenauer, "Parallelization and High-Performance Computing Enables Automated Statistical Inference of Multi-scale Models," *Cell Systems*, vol. 4, no. 2, pp. 194–206.e9, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405471216304124
[81] A.M., Osvaldo, R. Kumarand J. Lao, *Bayesian Modeling and Computation in Python*. Boca Raton, Florida, USA: Chapman & Hall, Dec 2021.
[82] E.M. Mirkes, J. Allohibi and A.N. Gorban, "Fractional Norms and Quasinorms Do Not Help to Overcome the Curse of Dimensionality," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary, 2019, pp. 1–8.
[83] C. Aggarwal, A. Hinneburg and D. Keim, "On the Surprising Behavior of Distance Metric in High-Dimensional Space," in *Database Theory — ICDT 2001*, J. Van den Bussche and V. Vianu, Victor, Ed. Heidelberg, Berlin, Germany: Springer Berlin Heidelberg, 2001, pp. 420–434.
[84] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970. [Online]. Available: https://doi.org/10.1093/biomet/57.1.97
[85] S. Duane, A.D. Kennedy, B.J. Pendleton and D. Roweth, "Hybrid Monte Carlo," *Physics Letters B*, vol. 195, no. 2, pp. 216–222, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037026938791197X
[86] T. Toni, D. Welch, N. Strelkowa, A. Ipsen and M.P.H. Stumpf, "Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems," *Journal of The Royal Society Interface*, vol. 6, pp. 187–202, 2009.
[87] N. Chopin , O. Papaspiliopoulos, *An Introduction to Sequential Monte Carlo*. Springer Cham, Oct. 2020.
[88] P.E. McKnight and J. Najab, *Mann-Whitney U Test*. John Wiley & Sons, Inc., 2010, pp. 1–1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470479216.corpsy0524
[89] J.D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*, 5th ed. New york, USA: Chapman and Hall/CRC, 2011.

MAHTAB MODABER is a PhD candidate in the Electronic Systems group at the Department of Electrical Engineering. Her research areas include performance modeling and analysis of complex systems. She has received her Master's degree in Electrical Engineering from Amir Kabir University and her Bachelor's degree in Electrical Engineering from Khaje-Nasir University ,Iran.

MARTIJN HENDRIKS received the M.Sc. (2002) and Ph.D. (2006) degrees in computing science from Radboud University Nijmegen, The Netherlands. Since 2022 he works as a researcher with the Department of Electrical Engineering at Eindhoven University of Technology, The Netherlands. From 2011 until 2022 he worked as a research fellow for the ESI group of the Dutch national institute for applied scientific research. His current interests include the modeling and analysis of embedded and cyber-physical systems, with an emphasis on performance engineering.

MARC GEILEN received the M.Sc. and Ph.D. degrees in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1996 and 2002. He is currently an Associate Professor with the Department of Electrical Engineering at Eindhoven University of Technology, Eindhoven, The Netherlands, and leading the Model-Based Design Lab. His research interests include model-based design processes, formal models-of-computation, design-automation algorithms, cyber-physical and networked embedded systems, and multi-objective optimization.

TWAN BASTEN (M'98-SM'06) received the M.Sc. and Ph.D. degrees in computing science from Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands. He is currently a Professor with the Department of Electrical Engineering, TU/e. He is also a Senior Research Fellow with ESI, TNO, Eindhoven. His current research interests include the design of embedded and cyber-physical systems, performance engineering, dependable computing, and computational models.

JEROEN VOETEN works as a full professor at the Eindhoven University of Technology and is the scientific director of the High Tech System Center. Between 2013 and 2019 he worked as senior scientist and was the scientific advisor of the ESI group of the Dutch national institute for applied scientific research. His professional passion is to improve industrial products and design processes through cutting edge model-based design methodologies, by working on the borders between academic research and industrial innovation. His research expertise is performance engineering, including the areas of (stochastic) performance analysis, design automation, scheduling and predictable synthesis. He worked actively in the application domains of telecommunication, consumer electronics and high-tech cyber-physical systems. He developed and supervised several multidisciplinary research programs that led to various industrial innovations and authored over hundred journal and conference publications.