A Case Study to Elicit Challenges for Performance Engineering of Cyber Physical Systems

Lorenzo Pagliari
Gran Sasso Science Institute
L'Aquila, Italy
lorenzo.pagliari@gssi.it

Raffaela Mirandola
Politecnico di Milano
Milano, Italy
raffaela.mirandola@polimi.it

Catia Trubiani
Gran Sasso Science Institute
L'Aquila, Italy
catia.trubiani@gssi.it

ABSTRACT

Cyber-physical Systems (CPS) are engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components. Therefore, the engineering of CPS is inherently collaborative, demanding cooperation between diverse disciplines. Besides, it requires attention to not only the functional aspects, such as behaviour and correctness, but also to performance characteristics, such as timing constraints of the entire system. In this paper we exploit an existing simulation environment to model and analyze a robotic system, and some preliminary performance analysis results are shown. This illustrative case study aims to bring upfront the challenges for performance engineering of CPS.

CCS Concepts

•Software and its engineering → Software performance;

Keywords

Performance Engineering; Cyber Physical Systems; Simulation Models

1. INTRODUCTION

Cyber-Physical Systems (CPS) represent an evolution of the traditional embedded systems, in fact they are known as the integration of computation with physical processes [9], or as an orchestration of computers and physical systems [12]. The overall idea is that the software part of CPS is able to understand the environment reading data from sensors and acting in the real world by interacting with actuators.

A well-know specification of CPS has been provided in [15] by the National Institute of Standards and Technology:

> "*Cyber physical systems are hybrid networked cyber and engineered physical elements co-designed to create adaptive and predictive systems for enhanced performance.* "

ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3053600.3053650

and the National Science Foundation posits that advances in CPS will enable systems with adaptability, scalability, resiliency, safety, security, and usability that will far exceed that of today's embedded systems [18, 15].

CPS, as well as embedded systems, have a wide range of application areas like health-care, medicine, power management, electric smart grid and renewal energy, automotive with smart car and intelligent road system with unmanned vehicle [1, 17]. Other examples of CPS can be found in the areas of smart buildings [20, 2], medical devices, military systems, traffic control and safety, process control, HVAC (i.e., heating, ventilation and air conditioning) systems, industry automation and manufacturing systems [13], air traffic control [12] and aircraft avionics systems [6, 12]. With respect to an embedded system, CPS are not unique integrated systems provided by an equipment manufacturer, on the contrary they are the composition of cyber and hardware components from different companies, delegating the choice of which component to adopt in the hands of engineers developing the CPS. Such broad set of application areas makes the task of designing CPS not trivial at all, in fact it is necessary to combine multiple components with different levels of abstraction along with their requirements [9, 6].

Due to all these factors, the development of CPS results to be a difficult task that is inherently collaborative, and requires demanding cooperation between diverse disciplines. Today there is still limited availability of technical solutions to deal with the modelling, analysis, and runtime support of CPS. In particular, modelling the integration of different system levels and their quality is still an open research problem, due to the interplay of different technical spaces. The goal of this paper is to elicit the challenges of this domain by presenting a case study modeled with an existing simulation environment, i.e., Ptolemy[1].

The paper is organized as follows. Section 2 provides an overview of state-of-art for performance engineering of CPS; Section 3 presents an illustrative case study to demonstrate that more research is needed in this context; Section 4 discusses the devised challenges and concludes the paper.

2. RELATED WORK

Several research challenges arise in the context of CPS [6, 15] and, as discussed in [15], the classical model-based design approach is not enough for modeling and analyzing CPS. Hereafter we discuss the methodologies that have been proposed in literature for this goal.

[1] http://ptolemy.eecs.berkeley.edu

Modeling of CPS. Modeling CPS involves different aspects of the design phase, e.g., both the hardware and behavioral part. This let to new modeling approaches to describe the new systems features and characteristics. In [6] the authors highlighted the need of new models of computation cooperating together for correctly modeling CPS. In particular, the authors focused on the actor-based model-integrated development approach commonly used for embedded systems. An interesting approach to model CPS is the *MechatronicUML*[3] method (that is based on the well-known UML standard), since it enables the model-driven design of discrete software of self-adaptive mechatronic systems. It is built on a component-based system model integrated with the controllers of the mechatronic systems, in order to verify safety properties. In [5] a model-driven development of a reconfigurable CPS system using the MechatronicUML formalism is presented. In [19] the problem of finding an adequate language that is able to describe the behavior of CPS is discussed, and a modeling language (namely *ACUMEN*) that supports hybrid (continuous/discrete) mathematical models is proposed. Another model-based approach for CPS is the co-simulation method that takes models from different specific domains and allows the simultaneous simulation and interaction between these different models. This method is suitable when the software, physical, and environment parts are described with different specific languages. In [11] a model-integrated development approach for real-time control systems is presented and it is integrated with a Mathlab-based simulator for real-time control system. In [7, 12] are presented and discussed the Ptides modeling approach that is a programming model serving as coordinator language for model-based design of distributed real-time systems. The Ptides models define the functional and temporal interaction of distributed software components, the networks that bind them together, sensors, actuators and physical dynamics. This approach is used in the Ptolemy tool. In [20] an approach extending the standard Building Information Models (BIM) is presented allowing the modelling of buildings, adding components that empower the specification and representation of the cyber-physical space. In [16] a resource based-sharing framework (RSBF) for CPS is presented. It is designed to model any CPS resource-sharing by combining elements from graph theory and social welfare theory, thus to describe complex systems, with the goal of maximizing the CPS overall utility maintaining a decentralized control. In [10] an approach to provide high assurances at runtime is presented as a distributed and adaptive real-time (DART) system, with a particular focus on the properties of such system, e.g., timeliness, resource constraints, etc.

Analysis of CPS. In [8] an existing simulation framework (FERAL) has been extended with the automotive open system architecture (AUTOSAR) standard that provides access to hardware services for the automotive field. FERAL is able to provide virtual evaluation platforms by coupling existing specialized simulators into one semantically integrated simulation scenario. It implements a similar concept of the co-simulation approach, as described in the modeling section. With the inclusion of AUTOSAR application in this framework, it is also possible for suppliers to perform virtual validation of automotive software without developing the specific required hardware, thus supporting the development paradigm shift from dedicated hardware-software systems to CPS.

Performance evaluation of CPS. In [14] the performance degradation of CPS is investigated by considering stealthy integrity attacks. The authors focus on quantifying the maximum perturbation that an attacker can introduce into a control system via a stealthy integrity attack in the feedback loop. Differently from our case study, in [14] the CPS performance is calculated with a linear stochastic model that does not include behavioural aspects of CPS. In [21] the problem of privacy and performance trade-offs in CPS is tackled. An optimization problem is formulated, and the system cost is optimized subject to privacy requirements. However, in [21] the performance evaluation is regulated by a control law including a restricted set of system parameters, whereas our case study includes a finer-grained level in the performance modelling of CPS. In [9] the problem of evaluating the performance of CPS is investigated by describing such systems as hybrid and dynamic distributed ones that are made of arbitrary compositions of timely (where execution time has known bounds) and untimely components. However, in [9] the simulation framework is sketched only, no case study is shown to demonstrate how the combination of multiple CPS models enables to get performance analysis results.

Summarizing, we found that in literature the performance evaluation of CPS is mainly supported by ad-hoc modelling and analytic/simulation tools, in fact the enormous variety of CPS makes the development of a uniform methodology too complex. Our case study focuses on the specification of multiple models to strengthen the level of abstraction for the derivation of performance metrics of interest.

3. CASE STUDY

Our illustrative case study is a cleaning robot (CR) that moves through rooms in a building to clean up dirtiness. The scenario is presented in Section 3.1, and with the support of the Ptolemy II tool we define multiple models for triggering its performance engineering, as described in Section 3.2. The tool is constituted by a simulation environment from which we get some preliminary experimental results that are shown in Section 3.3.

3.1 Scenario description

The application scenario is inspired by the work of Bures et al. in [4]. In particular, a cleaning robot, potentially a set of robots, has to clean up a room, ideally a two dimensional space. The robot is equipped with an electric engine that allows it to move around the room, a downwards-looking camera and sensors that let it to identify dirtiness on the floor and for sake of simplicity each event of dirtiness can be associated to a dirty tile in the room. The cleaning robot has its own battery and when the energy level decreases under a specified threshold, the robot goes toward a charging station to recharge its battery. In Figure 1 we report an illustrative scenario for our case study by depicting the room as a grid where the robot moves to clean dirty cells, and a charging station is located in one cell of the room.

The robot starts at some cell of the grid and it looks for dirtiness around its location. The dirty spots are randomly distributed in the room, i.e., it is not predefined when the robot starts its cleaning duty. To be more realistic, each dirty spot is different in terms of how much is dirty, thus the time spent by the robot to clean up a dirty tile is different for each dirty tile. Moreover, if the robot is looking for dirtiness from a while but it does not find any, then the

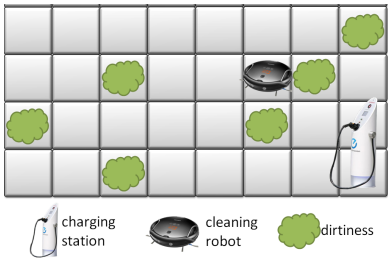


Figure 1: Case study scenario (inspired by [4]).

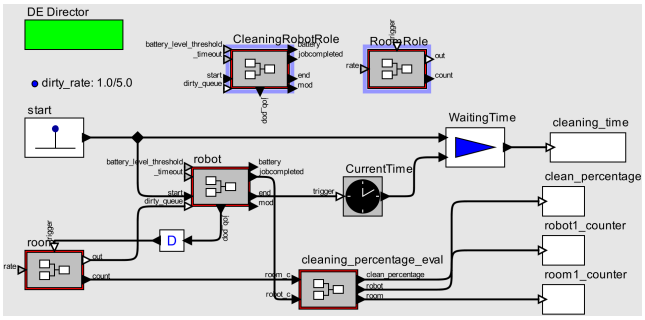


Figure 2: Scenario top-level model.

robot assumes the room is clean enough and it stops, coming back to the ground station. This choice has been made to simplify the model, but as future work we want to extend the model by considering more rooms, hence instead of stopping, the robot can move to an adjacent room. A robot that is moving around or is cleaning, consumes energy and when the robot battery level drops under a certain threshold, the robot concludes any in progress operation and then goes to the charging station. When the robot's battery is recharged over a certain threshold, the robot becomes operative again, coming back searching for any other dirty spot.

3.2 Scenario models

In this section we describe some preliminary models specific for our case study with the goal to enable its performance engineering. Figure 2 reports the top level model of our scenario. This model includes two main entities, i.e., the cleaning robot and the room.

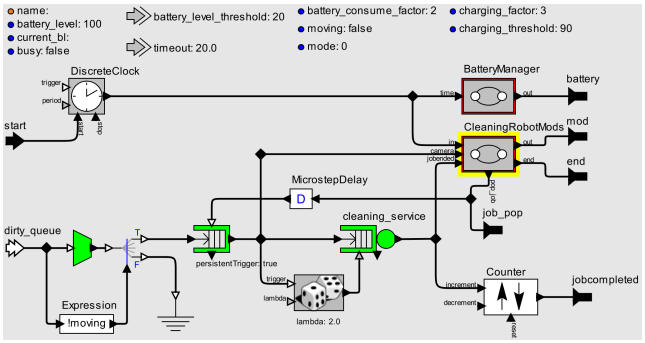
Our model is constituted of one instance of the *RoomRole* class (that is named *room*) and one instance of *CleaningRobotRole* class (that is named *robot*). The whole model is managed by a Discrete Event (DE) Director through which it is possible to specify the start and stop times for the simulation. On the left side of Figure 2 there is a *Single Event* actor called *start* that emits a single signal at time zero and initiates the robot. The room instance generates randomly dirty events that send to the robot through the connection from its output port to the dirty queue input port of the robot. Each time the robot fully clean a dirty spot, requires any new dirty event to the room, through the loop back link on the room trigger port. When the robot decide that the room is clean enough and stops, emits a *end* signal that triggers the measurement of the cleaning time. The *WaitingTime* actor takes in input two events and outputs the time elapsed between them. On the bottom of the model, there is a component named *cleaning percentage evaluator* that takes as input the counted dirty events created by the room and the counted cleaned events by the robot, thus to compute the percentage evaluation. In this way we are able to derive after each run the cleaning percentage and the time spent by the robot to reach that percentage. To enable a parametric evaluation, we introduced the *dirty_rate* parameter that represents how often dirty events are generated in the room, but this parameter can also be read as the speed at which the robot is moving. Similarly, for the robot class, parameters *battery_level_threshold* and *_timeout* are set with specific value but its possible to change them at run time, by passing new values directly to the respectively input ports.

For sake of space, we omit the description of the Room Model and we describe hereafter the Robot Model only shown in Figure 3. Figure 3(a) shows how all the domains and the system environment are represented in the robot model, the state domain model (battery management) is shown in Figure 3(b), whereas the behavioral domain model (cleaning robot management) is shown in Figure 3(c).

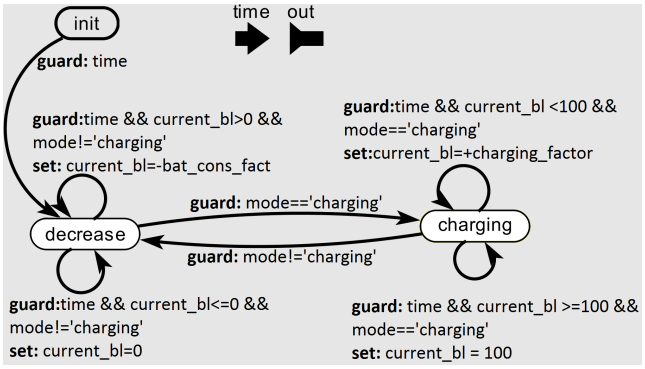
Figure 3(a) reports the parameters that characterize the robot: the *name*, the *battery_level* is the energy level with which the robot starts working, the *current_bl* is the current battery level and it is used to evaluate the energy dynamics, the *busy* parameter is used in the mode controller and signals if the robot is in a busy state. There are two PortParameters: the *battery_level_threshold* that is the energy threshold under which the robot goes to a charging station and the timeout represents the time that has to elapse without any dirty event before the robot decides to stop cleaning. Moreover, the *battery_consume_factor* parameter represents the energy consumption at each time step, and the *moving* parameter indicates if the robot is moving from the operative state to the stop state; finally the *mode* parameter represents with an integer number the operation modes of the robot and it is used as synchronization between the control actor. The last two parameters are: (i) the *charging_factor* that is the dual of the *battery_consume_factor* and represents the energy charging factor of the battery; (ii) the *charging_threshold* that is the dual of the *battery_level_threshold* but for the charging phase.

Figure 3(b) shows the structural domain model of the BatteryManager controller component as a finite state automaton. The controller starts in the initial state *init* and after receiving a signal on the time port, it goes to the decrease state until the robot goes into a charging mode (i.e., mode==30), i.e., the guard transition *mode=='charging'*. Until that, it continues to decrease the battery by the *battery_consume_factor* (*bat_cons_fact* in the transition) parameter at each time step. When the battery reaches zero it keeps such value. When the robot switches to charging mode, the battery controller goes into the charging state, where it will increase the current battery level by the *charging_factor* at each time step. When the battery reaches the maximum it will stabilize the level to the maximum. When the robot switches back to an operational mode, then the battery manager goes again into the decrease state.

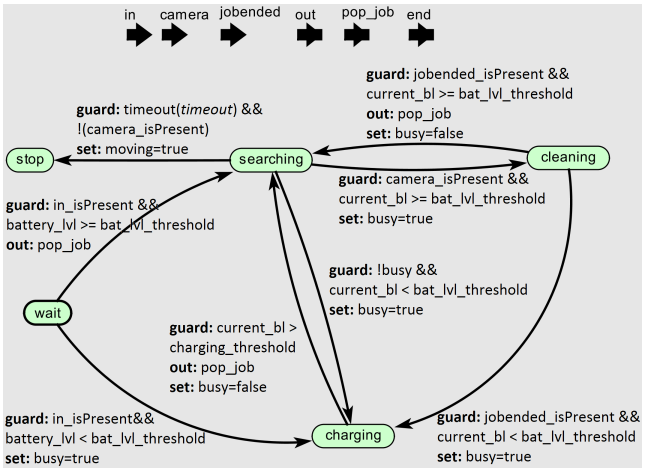
Figure 3(c) depicts the cleaning robot modal model. The mode controller is the robot brain and represents the control software that manages the robot operational modes. The controller starts on the initial state *wait* and when there is a signal on the input port *in*, it goes into the searching mode if the initial battery level passed as parameter is greater than the battery threshold, otherwise it goes into the charging



(a) Robot model.



(b) Battery Manager model.



(c) Cleaning modes model.

Figure 3: Robot model and behavioral models.

mode. If it is in the charging mode, when the current battery level will be greater or equal to the charging threshold, it moves to the searching mode, calling for a job. When the robot is looking for a job, it waits for a trigger of the *camera* input port that signals if a dirty event is detected by the floor camera. Meanwhile it is searching, if no dirty event is triggered and *_timeout* time is elapsed, then it makes the assumption that the room is cleaned, and it stops moving, i.e., it goes into the stop mode and emits the *end* signal. On the contrary, if there is a signal on the *camera* input port before the timeout expires then the controller moves to the cleaning mode, i.e., the robot is busy cleaning a tile. As shown in Figure 3(c), it keeps the cleaning state until the service station completes the job and emits a signal on the controller *jobcompleted* input port. When there is a signal on that port, it means that the dirtiness was cleaned and the system returns back to the searching state requesting a new job if there is enough battery.

3.3 Experimental results

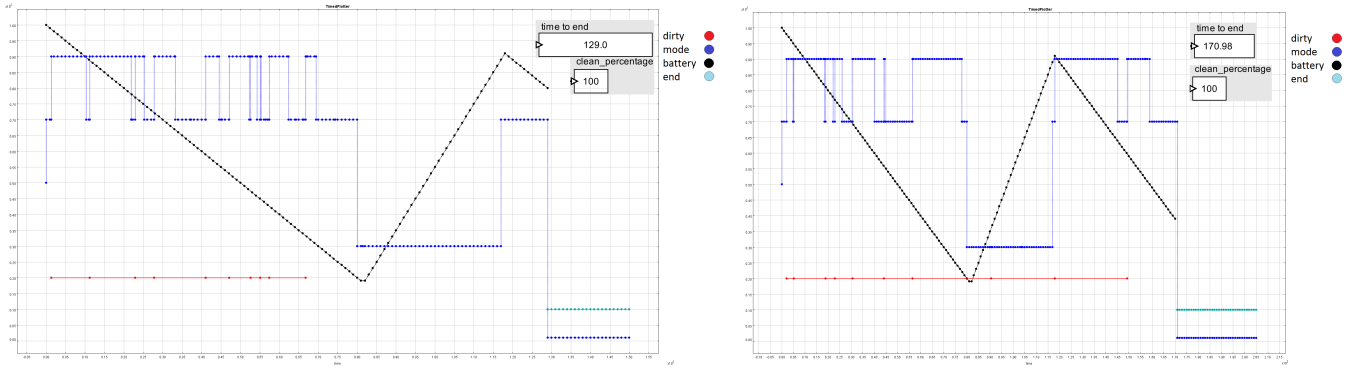
For our case study we measured the following indices: (i) *cleaning time*, i.e., the time spent from the robot to clean a room, calculated from when it is started until it assumes the room is cleaned; (ii) *clean percentage*, i.e., the number of dirtiness events that have been cleaned over the total number of dirtiness events generated by the room.

In the following we report some plots obtained from simulating the models and varying different parameters. All the simulation runs were made with a total number of generated dirty events of ten ($dirty\_amount = 10$), a battery level threshold equal to 20% under which the robot needs charging ($battery\_level\_threshold$=20%), and a charging threshold equal to 90% over which the robot starts again to clean ($charging\_threshold$=90%). The plots are generated by showing the battery level with a black line, the robot modes with a blue line and fixed values (i.e., wait=50, searching=70, cleaning=90, charging=30, stop=0), the dirty events are shown with red dots, and the end signal with a light blue.

Figure 4 shows the results we obtained while fixing the dirty generation rate and the robot timeout, but varying the robot service time, in fact the used parameters are: (i) $dirty\_rate = 0.2$ and (ii) $timeout = 12s$, and (iii) $lambda = [0.1, 0.2]$, i.e., the *service time* of the cleaning robot is varied with two values. Figure 4(a) depicts the robot service time with an exponential distribution of $lamda = 0.2$ so the mean time of cleaning a dirty event is 5 seconds. As the figure shows, the robot is able to clean all the dirty events before the timeout expires and the robot battery level goes under the threshold. The robot goes into the recharge state and after overing the charging threshold, it waits again for the timeout to expire. Then it stops by concluding its job in 129 seconds and achieving a cleaning percentage of 100%. In Figure 4(b) instead we increase the lambda to 0.1, slowing down the service time that in average is of 10 seconds. Also in this run the robot, after a charging phase, is able to complete all the jobs, but with an higher total time of 170.98 seconds. As expected, changing the service time value distribution affects the total time spent by the robot.

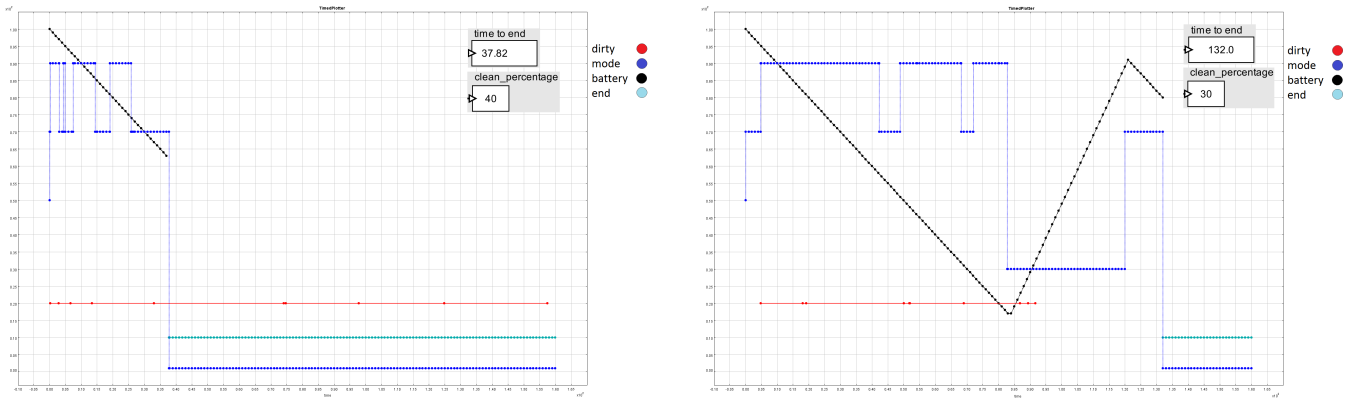
Figure 5 reports the results obtained while changing the dirty generate rate from 0.2 to 0.1, i.e., slowing it down, whereas the timeout is unchanged. We can notice that with a lower generation rate, the timeout we fixed is not enough to let the robot completing all the jobs. Figure 5(a) shows that the robot is able to clean the first four dirty events but then the timeout expires and the robot stopped after 37.82 seconds achieving a clean percentage of 40%. Similarly to Figure 4, in Figure 5(b) we slow down the service time of the robot and this change has the same effect as in the previous experiment, and we can notice that with this change the robot reaches a 30% cleaning percentage in 132 seconds.

Summarizing, all these plots aim to demonstrate the importance of system parameters and how they affect the CPS performance indices. Further challenges come from the specification of spatial characteristics, e.g., the size and the spatial position of jobs in a room. Besides, the *dirty_rate* pa-



(a) $lambda = 0.2$ (b) $lambda = 0.1$

Figure 4: Plots for $dirty\_rate = 0.2$ and $timeout = 12s$, but different service time.



(a) $lambda = 0.2$ (b) $lambda = 0.1$

Figure 5: Plots for $dirty\_rate = 0.1$ and $timeout = 12s$, but different service times.

rameter can also represent the speed at which the robot is moving, i.e., a faster robot detects possible dirty spots more frequently since the generation rate of dirty jobs is higher. On the contrary, a slower robot encounters possible dirty spots less frequently since the generation rate of dirty jobs results to be lower. We also added the *dirty_tiles* parameter that represents the amount of dirtiness in the room, but also its size. A change in this parameter means either a more dirty or a changed size room.

4. DISCUSSION AND CONCLUSION

During the development of the case study we have identified several challenges related to the research area of performance engineering of CPS. For some of them we have started a deep investigation, while some others result still open. In the following, we briefly summarize the most relevant challenges:

- *Requirements*: a well defined expression of requirements for this type of systems, where different domains have to coexist, can be difficult. Indeed, different parts of the system have multiple performance attributes that must be evaluated, each one with its own unity of measure, scale and precision. How to express these attributes into requirements is still an open problem. In our case study we focused on the cleaning percentage and the relative time spent to reach that coverage by the robot. We have then defined appropriate parameters and aggregated measures, however a formal approach would be necessary to guide the CPS development process.

- *Modeling*: one of the major challenges is how to model CPS. It is indeed difficult to define an hybrid formalism that allows the combined modelling of software and hardware, and to take also into account performance characteristics. In literature the adopted solution is often to use different models, and then let them coexist during all the system life cycle. In our example, adopting Ptolemy, we used finite state machines and modal models for the software and the behavioral parts, and component models for the hardware and the environment. However, a well defined (possible) model-driven approach for the different model definitions and interactions is still missing.

- *Uncertainties*: due to the nature of CPS, the knowledge of the environment is not complete nor accurate. This entails that the information included in the models is subject to uncertainties. In this area we devise two main challenges: (i) how to explicitly address the uncertainty at requirements level, and (ii) how to recognize its presence at model level and how to manage it, thus to mitigate its potentially negative effects and increase the level of assurance in a given CPS. In the presented case study, we modeled the environ-

ment uncertainty with the random generation of dirtiness events in the room. However, a complete definition of models and methods for defining, recognizing and taming uncertainties is still missing.

- *Analysis and performance metrics*: the enormous variety of CPS makes the specification of performance metrics to analyse very difficult, in fact it is difficult to select the relevant metrics among the stadard ones, such as reponse time, utilization, and throughput. Moreover, the combined use of multiple models at different abstraction levels leads to possible problems in their performance analysis. It is indeed important to take into account the different aspects and their combination still maintaining, when possible, a separation of concerns among the various parts of the systems. In the modeled case study, we have defined an hybrid model that we solved with the simulation environment of Ptolemy. The model analysis allowed us to quantify the overall system performance and the impact of parameter changes. However, at present, the model analysis part is not predefined and it strongly depends on the designer skill in the correct definition of necessary measurements and involved components. A guided process allowing also the separation of concerns is still missing.

-*Refactoring*: the usage of models in the system design phase allows, among the other advantages, the possibility of design space exploration. This facility is key in the definition of possible refactoring actions when performance requirements are not met, for example. A challenge here is represented by the definition of suitable parameterizable models. In the considered case study, we did a large use of parameters and, as discussed in Section 3, we analyzed how the change of value or meaning of these parameters leads to different models and results. In this way, comparing the results collected from different model settings allows the comparison of a wider set of design choices. A well defined methodology for design space exploration is still missing and would be of key relevance in the design of CPS.

In the future, we plan to move along the research lines we have listed above. According to that, we plan to investigate other case studies, possibly from the industrial contexts, thus to extract further challenges of this domain. Besides, we are working on developing a multi-modelling approach that exploits both the principles of model-driven engineering and the knowledge acquired in the closed domain of hardware-software co-design. The vision of such approach is to enable the performance evaluation of CPS by building models able to provide design alternatives dealing with performance requirements.

5. REFERENCES

[1] R. Baheti and H. Gill. Cyber-physical systems. *The impact of control technology*, 12:161–166, 2011.

[2] B. Balaji and al. Models, abstractions, and architectures: the missing links in cyber-physical systems. In *Annual Design Automation Conference*, page 82. ACM, 2015.

[3] S. Becker and al. The mechatronicuml design method–process, syntax, and semantics. *SE group, Heinz Nixdorf Institute, University of Paderborn, Tech. Rep. tr-ri-12-326*, 2012.

[4] T. Bures and al. Statistical approach to architecture modes in smart cyber physical systems. In *IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 168–177, 2016.

[5] S. Burmester, H. Giese, and M. Tichy. Model-driven development of reconfigurable mechatronic systems with mechatronicUML. In *Model Driven Architecture*, pages 47–61. Springer, 2005.

[6] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *IEEE*, 100(1):13–28, 2012.

[7] J. C. Eidson and al. Distributed real-time software for cyber–physical systems. *IEEE*, 100(1):45–59, 2012.

[8] P. Feth, T. Bauer, and T. Kuhn. Virtual validation of cyber physical systems. In *Software Engineering & Management*, pages 201–206, 2015.

[9] A. E. S. Freitas and R. M. da Silva Bezerra. Performance evaluation of cyber-physical systems. *ICIC Express Letters*, 10, 2016.

[10] S. A. Hissam and al. Engineering high assurance distributed cyber physical systems. Technical report, DTIC Document, 2015.

[11] G. Karsai and J. Sztipanovits. Model-integrated development of cyber-physical systems. In *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pages 46–54. Springer, 2008.

[12] E. A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015.

[13] J. Lee, B. Bagheri, and H.-A. Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[14] Y. Mo and B. Sinopoli. On the performance degradation of cyber-physical systems under stealthy integrity attacks. *IEEE Trans. Automat. Contr.*, 61(9):2618–2624, 2016.

[15] P. J. Mosterman and J. Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(1):5–16, 2016.

[16] A. Nayak and al. Resource sharing in cyber-physical systems: modelling framework and case studies. *International Journal of Production Research*, 54(23):6969–6983, 2016.

[17] J. Shi, J. Wan, H. Yan, and H. Suo. A survey of cyber-physical systems. In *International Conference on Wireless Communications & Signal Processing WCSP*, pages 1–6, 2011.

[18] Steering Committee for Foundations in Innovation for Cyber-Physical Systems. Strategic opportunities for the 21st century cyber-physical systems, 2013.

[19] W. Taha and R. Philippsen. Modeling basic aspects of cyber-physical systems. *arXiv preprint arXiv:1303.2792*, 2013.

[20] C. Tsigkanos and al. Adding static and dynamic semantics to building information models. In *Proceedings of the International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 1–7. ACM, 2016.

[21] H. Zhang, Y. Shu, P. Cheng, and J. Chen. Privacy and performance trade-off in cyber-physical systems. *IEEE Network*, 30(2), 2016.