Model-based Performance Analysis for Architecting Cyber-Physical Dynamic Spaces

Riccardo Pinciroli
Gran Sasso Science Institute
L'Aquila, Italy
riccardo.pinciroli@gssi.it

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

***Abstract*—Architecting Cyber-Physical Systems is not trivial since their intrinsic nature of mixing software and hardware components poses several challenges, especially when the physical space is subject to dynamic changes, e.g., paths of robots suddenly not feasible due to objects occupying transit areas or doors being closed with a high probability. This paper provides a quantitative evaluation of different architectural patterns that can be used for cyber-physical systems to understand which patterns are more suitable under some peculiar characteristics of dynamic spaces, e.g., frequency of obstacles in paths. We use stochastic performance models to evaluate architectural patterns, and we specify the dynamic aspects of the physical space as probability values. This way, we aim to support software architects with quantitative results indicating how different design patterns affect some metrics of interest, e.g., the system response time. Experiments show that there is no unique architectural pattern suitable to cope with all the dynamic characteristics of physical spaces. Each architecture differently contributes when varying the physical space, and it is indeed beneficial to switch among multiple patterns for an optimal solution.**

***Index Terms*—Cyber-Physical Systems, Dynamic Physical Space, Software Performance Engineering.**

I. Introduction

Cyber-Physical Systems (CPS) have been defined as an evolution of embedded systems, mainly differentiating for the interplay between software and hardware components that inevitably triggers new challenges [1], [2]. When focusing on software architectures [3]–[5], CPS resulted to attract the attention of the research community, and several methodologies recently emerged to deal with this domain [6]–[9]. However, to the best of our knowledge, only a few approaches provide a quantitative evaluation when architecting CPS. For instance, in [10] a framework is proposed for simulating CPS to derive some metrics of interest (e.g., energy consumption). This experience is later exploited for the Internet of Things (IoT) domain in [11], where probabilistic model checking is adopted to verify Quality-of-Service (QoS) requirements.

In this paper, we focus on the model-based performance analysis of CPS. We aim to investigate if software performance engineering techniques [12] can efficiently support software architects in the task of specifying the most suitable (from a performance perspective) design alternative. The target domain is represented by the physical space subject to dynamic changes, and we are interested to understand to what extent changes in the operational environment impact the CPS under analysis. For example, let us consider that there are some objects (e.g., a tray transporting medicines in a hospital) moving and occupying transit areas. This might disable the feasibility of some paths for automated machines that were supposed to cross such areas. Another scenario is represented by doors and windows that are closed or open with a certain probability. This affects the time required by robots to deliver goods or by drones to extinguish a fire, e.g., within a certain building. All these scenarios share the management of probabilistic parameters in the specification of the physical space, and this may largely affect the choice of the underlying software architecture. In the literature, the evolution of cyber-physical spaces is tackled by some approaches, e.g., [13]–[15], however, the verification of these systems mainly consists of reachability properties, whereas aspects related to software architectural patterns are neglected. The novelty of this paper relies on embedding architectural alternatives as part of the problem specification, to investigate the most suitable design alternatives under different space dynamics.

Starting from the specification of multiple architectural patterns that enable the self-adaption of CPS [16], we build performance models that are representative of the architectural alternatives. This way, we are interested to provide a quantitative evaluation of the feasible architectural alternatives, so that it is possible to early identify (and prevent) performance issues in CPS. We consider three different architectural patterns: (i) *centralized* pattern in which all the cyber entities communicate the status of the physical space with a central coordinator; (ii) *semi-decentralized* pattern which introduces a set of cyber entities acting as local coordinators for a subgroup of other entities, most likely those occupying spaces adjacent to the physical changes under analysis; (iii) *fully-decentralized* pattern in which each cyber entity is in charge of verifying the status of the space, and it is autonomous in the process of making decisions. Our experiments show that there is not a unique architectural pattern suitable to cope with performance-related requirements, pros and cons arise for all of them, since peculiarities of the space come into play.

Summarizing, the contributions of this paper are: (i) the specification of performance models expressing the peculiarities of cyber-physical dynamic spaces; (ii) the modelling and the analysis of different architectural patterns and their experimentation on realistic scenarios; (iii) empirical evidence on the benefit of early identifying (and prevent) performance issues for systems subject to dynamic cyber-physical spaces. Performance models and replication data are publicly available at the following link: https://doi.org/10.5281/zenodo.4493760.

II. Motivating scenario

In this section, we introduce a smart hospital case study as a motivating scenario, and we use it as a running example of a cyber-physical dynamic space throughout the paper. It is inspired by the recent trend of designing intelligent systems for tracking and monitoring COVID-19 patients, as well as smart sanitizing [1]. For instance, autonomous robots can help in some healthcare systems, such as virtual clinics, smart guard, and in providing medicines, thermometers, disinfectants, and cleaning supplies. Besides, robots can sanitize rooms by traversing physical spaces hosting patients with diseases, and using random path planning algorithms.

We start with a brief description of the static structure of the cyber-physical space and then consider its dynamics, i.e., possible ways in which space may change over time. Our interest is in understanding the impact of dynamic spaces on the performance evaluation of different architectural patterns, thus comparing these patterns on the overall performance of the system, e.g., the response time.

The cyber-physical space consists of a hospital environment with corridors, rooms, doors, stairs, and elevators. Corridors are used to reach rooms that may be connected through doors, which are either locked or unlocked. Stairs are used to move between different floors, similarly to elevators, however let us assume that only *two-legged* robots are able to take stairs, while all of them (i.e., including *wheeled* robots) can use elevators. A graphical representation of the scenario under analysis is depicted in Figure 1, where some of the provided services are listed at the top, whereas the bottom part shows the cyber-physical space with its main constituent elements.

The possible changes in the cyber-physical space that constitute the dynamics of our scenario are regulated by the probabilities on the status of the physical objects, specifically:

- *doors*, i.e., the probability of a door to be open or closed that can be determined by considering the type of involved rooms, e.g., surgeries are usually connected through doors that are closed most of the time;
- *stairs*, i.e., the probability that robots can take the stairs, it can be determined by considering if such transit areas are already occupied, e.g., stairs to the kitchen areas are most likely to be crowded around lunch/dinner times;
- *elevators*, i.e., the probability of taking an elevator, and it can be determined by considering if there is enough space for a robot, e.g., there might be some elevators usually used for stretchers with a reduced capacity.

We foresee some performance overhead due to the necessity of changing paths when doors or stairs are not available to reach the destination of the robot. Moreover, we consider some waiting time in the case of elevators showing a low capacity.

[1] https://arxiv.org/pdf/2007.10477

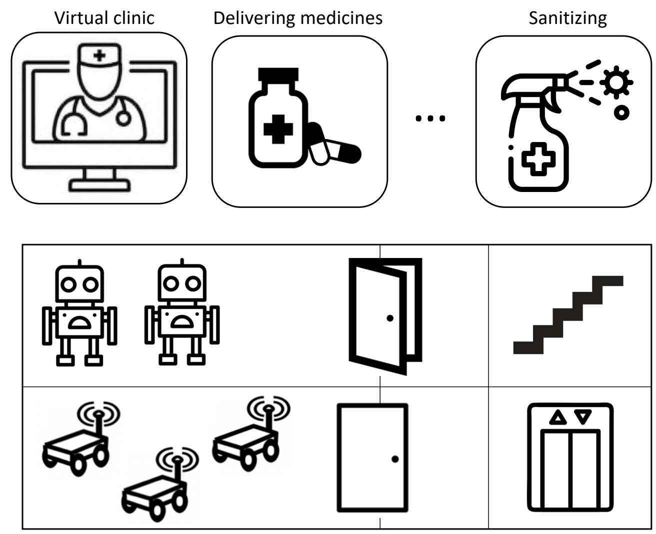
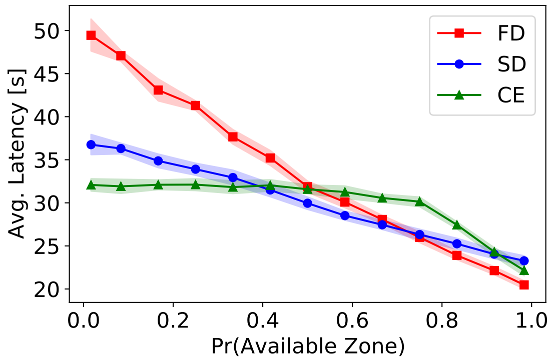


Fig. 1: Smart Hospital as Cyber-Physical Dynamic Space

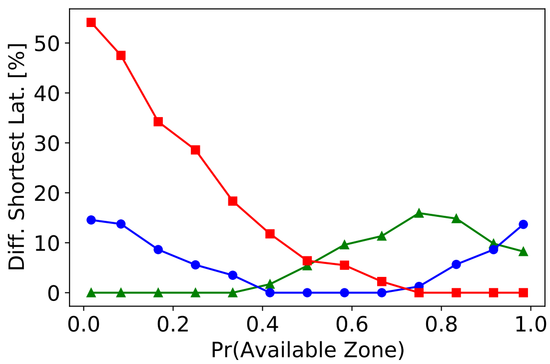
Besides the static structure and dynamics presented, let us consider a performance requirement that needs to be fulfilled by the design of a smart hospital environment: *Robots must be able to deliver medicines within 40 seconds.* This requirement triggers our model-based performance analysis that is in charge of evaluating different architectural patterns.

For illustration purposes, we conducted a preliminary study by manually building a performance model for robots moving between two areas and crossing a door with a variable probability of being open. The results are presented in an informal manner; a concrete model instance, its formalization, and its parametrization will be presented in the following sections. The description of the performance model input parameters is given in Table II, and their numerical values (derived by exploring some literature in this domain [17], [18]) are reported in Table III. It is worth remarking that further numerical values can be used, e.g., latencies and probabilities can be deduced by adopting ad-hoc simulation environments.

Figure 2a shows the average latency, expressed in seconds (see y-axis), of delivering medicines while considering one hundred robots moving through different corridors and crossing one door only. The probability of such a door to be open is reported on the x-axis. Curves are representative of the considered architectural patterns that are: (i) central (CE), i.e., robots communicate the status of the door to a central coordinator unit; (ii) semi-decentralized (SD), i.e., robots communicate if the door is open/closed to all other robots occupying the same room; (iii) fully-decentralized (FD), i.e., robots do not communicate the status of the door, they autonomously verify it. We can notice that when the door is always closed, i.e., $Pr(AvailableZone) = 0$, the FD pattern is the worst one showing a response time of 50 seconds roughly. It does not fulfill the stated requirement, and this is due to the overhead on recalculating the path that is paid by all robots. As opposite, when the door is always open, i.e., $Pr(AvailableZone) = 1$, this pattern becomes the best



(a) Average Latency



(b) Distance from the shortest latency

Fig. 2: Different architectural patterns acting in the system while varying the probability of a door being open.

architectural solution since robots are faster and do not pay any communication cost. The CE architectural pattern results to be the best one when the door has a high probability of being closed, whereas it turns to be less efficient when the probability is larger than 0.5 since the SD architectural pattern is instead more convenient. Figure 2b better visualizes that there is no unique architectural pattern overcoming all the others in the considered scenario. To make it clearer, on the y-axis we depict the difference with the shortest latency (among the three patterns) expressed in percentage values. We can notice that the CE architectural pattern is the best up to a probability of 0.4, then the SD pattern becomes the optimal solution up to a probability of 0.7 roughly, and later the FD pattern shows the shortest latency values.

This preliminary analysis motivates us to further investigate the problem. We develop a method to support software architects in the task of analyzing the performance of three architectural patterns for cyber-physical dynamic spaces.

III. Our Method

In this section, we present the performance models that have been built for dealing with cyber-physical dynamic spaces. Although different performance modelling formalisms [19] can be adopted to investigate this problem (e.g., queuing networks, continuous-time Markov chains [20]), we make use of Generalized Stochastic Petri Nets (GSPN) [21] since they have been recently applied to many domains, e.g., blockchains [22], edge-computing [23], and cyber-physical systems [24]. Further reasons to select GSPN are: (i) it is a formal method that allows avoiding ambiguity, (ii) its graphical notation is easy to understand, and (iii) there exist numerous tools that can solve and simulate GSPN-based models to derive performance metrics of interest, such as system response time. A known limitation of GSPN is the state space explosion, in fact simulation time may be affected by the size of scenarios under analysis [21]. We assume that the performance analysis is conducted at design time, and results are reported to support software architects in comparing different design alternatives.

A. *Performance modelling of Dynamic Spaces*

Let us start considering the dynamics affecting our motivating scenario, i.e., robots delivering medicines. We identify two main types of dynamics that are handled in different ways. Specifically, there are zones that show (i) *temporary* dynamics, such as closed doors, overcrowded stairs, or physical objects occupying transit areas; (ii) *permanent* dynamics, e.g., an elevator has a limited capacity, it accepts a finite number of robots, and the remaining ones need to wait before using it. In the former case, we foresee robots deciding to change their path and look for alternatives, whereas in the latter case we envisage robots waiting to access. As said in Section II, we consider two different types of robots, and for wheeled robots, the elevator is the only way to move among different floors.

The case of temporarily unavailable zones can be abstracted as resources that are (un)available with certain frequency overtime, i.e., a probability can be associated to denote the (un)availability of different zones. Figure 3 depicts the reference performance model; it consists of two places denoted as circles (i.e., *unavailable* and *available*) and two timed transitions represented as rectangles (i.e., *switch-U-A* and *switch-A-U*). Places determine the state of the zones that can be either unavailable or available, whereas transitions are in charge of capturing the events leading to migration from a state to another. A token (depicted as a small black circle inside a place) means that the corresponding system state holds, e.g., in Figure 3 our assumption is that the zone has an initial state of being unavailable. The *switch-U-A* transition follows an exponential distribution with average time $\mu_{switch-U-A}$ that specifies how often the zone goes from *unavailable* to *available* states. Similarly, $\mu_{switch-A-U}$ is the average time that the zone is available, and the firing of that transition captures the time required by the corresponding zone to change its state from *available* to *unavailable*. Hence, the probability that a zone is available is computed as $\mu_{switch-A-U}/(\mu_{switch-A-U}+\mu_{switch-U-A})$.

Figure 4 reports the performance model for zones subject to permanent constraints. Let us assume the elevator can handle up to K robots at a time, and this is reflected in the place named *capacity* showing K tokens. Places *boarding-L* and *boarding-U* denote the multiplicity of robots (i.e., the number of tokens inside the places are N and M) ready for boarding at the lower (L) and upper (U) floors, respectively. For readability

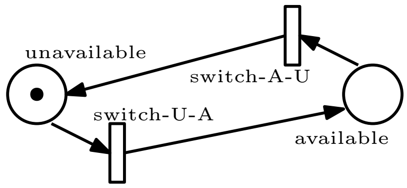


Fig. 3: Performance model for temporarily (un)available zones.

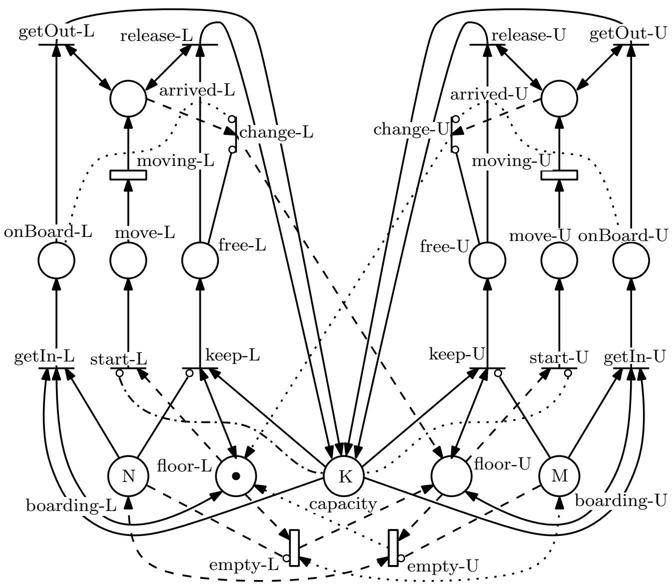


Fig. 4: Performance model for zones showing permanent constraints. Arrows show a different dash style for readability.

reasons, Figure 4 reports N and M parameters that represent the aggregated values on the two types (i.e., wheeled and two-legged) of robots, e.g., $N = N_{wheeled} + N_{two-legged}$. Places *floor-L* and *floor-U* model the state of the elevator, and our assumption is that initially it is on the lower floor, i.e., one token is showed in *floor-L*. The timed transition *empty-L* (or *empty-U*) enables the change of floor (whose average time is $\mu_{empty-L}$) in case the place *boarding-L* (or *boarding-U*) is empty. This way, the elevator can move even if it is empty to reach a floor where robots are waiting to be served.

If $N \geq K$, then K robots migrate to *onBoard-L* place through the activation of the *getIn-L* immediate transition (represented as a thin line in Figure 4), i.e., no time is associated to that operation. Consequently, the *capacity* place becomes empty, and the *start-L* immediate transition is enabled. Then, the *move-L* place gets one token, and after the time established for the *moving-L* timed transition, such a token is absorbed from *move-L* and generated in *arrived-L*, i.e., expressing the elevator moves from the lower to the upper floor. Note that moving a single token from *move-L* to *arrived-L* allows modelling the elevator (and all the robots inside it) reaching its destination in a time *moving-L*. Such a time is then used to compute the time required by each robot to complete the delivery. At this point, robots get out (see *getOut-L* immediate transition) from the elevator, and the capacity is put back to its original setting. The

TABLE I: Description of parameters that can be configured for the performance modelling of dynamic spaces.

Dynamics	Parameter	Description
Temporary	$\mu_{switch-U-A}$	average time the zone is unavailable
	$\mu_{switch-A-U}$	average time the zone is available
Permanent	$N_{wheeled}$	number of wheeled robots in the lower floor
	$N_{two-legged}$	number of two-legged robots in the lower floor
	K	capacity of the elevator
	$M_{wheeled}$	number of wheeled robots in the upper floor
	$M_{two-legged}$	number of two-legged robots in the upper floor
	$\mu_{empty-L}$	average time required by the elevator, when empty, to move from the lower to the upper floor
	$\mu_{empty-U}$	average time required by the elevator, when empty, to move from the upper to the lower floor
	$\mu_{moving-L}$	average time required by the elevator to carry robots from the lower to the upper floor
	$\mu_{moving-U}$	average time required by the elevator to carry robots from the upper to the lower floor

status of the elevator (i.e., from lower to upper floor) is updated through the *change-L* immediate transition that generates one token in *floor-U* place. If $N < K$, then N tokens are still moved to the *onBoard-L* place as in the previous case and $(K - N)$ tokens are moved to *free-L* place through the *keep-L* immediate transition. This way, the model keeps track of the number of empty spots in the elevator. After the elevator reaches the upper floor, its capacity is reconfigured again through the *getOut-L* and the *release-L* immediate transitions. The same procedure holds for the elevator moving from the upper to the lower floor, and showed in the rightmost part of Figure 4.

Table I reports a brief description of all the parameters that are defined for the performance modelling of cyber-physical spaces subject to temporary and permanent dynamics. The numerical value of these parameters can be set by software architects that are interested to make use of our models for the performance analysis of their application scenarios.

GSPNs are suitable to model system concurrency, and the model of a zone can be easily replicated for multiple doors, stairs, elevators, and various objects occupying transit areas. In the case of multiple zones, we can distinguish them by assigning a progressive number, e.g., a scenario with three doors will be analyzed as part of our experimental evaluation in Section IV. Each zone can be regulated by its own parameters, e.g., two doors may show a different probability of being available.

B. Performance modelling of Architectural Patterns

In this section, we describe the GSPN-based models built to investigate the performance of three architectural patterns [16]. These models include a set of parameters described in the sequel and summarized in Table II. When evaluating the performance characteristics of the architectural patterns, soft-

TABLE II: Description of parameters that can be configured for the performance modelling of architectural patterns.

Pattern	Parameter	Description
All	N	number of robots
	$\mu_{switch-U-A}$	average time the zone is unavailable
	$\mu_{switch-A-U}$	average time the zone is available
	μ_{wait}	average time that robots wait to receive a task
	μ_{reach}	average time for robots to reach an obstacle
	$\mu_{goStraight}$	average time for robots to go straight in their target path
	μ_{turn}	average time for robots to turn and go back
	$\mu_{goAround}$	average time for robots to go around the obstacle
SD	μ_{follow}	average time spent to communicate with the robot spreading the notice
CE	μ_{fail}	average time required for the communication between the robot that notifies the presence of an obstacle to the central coordinator
	μ_{ask}	average time required by robots to ask information about the status of a zone to the central coordinator
	$\mu_{refresh}$	average time for triggering the control on the (un)availability of zones.

TABLE III: Input parameters used to obtain the results of our preliminary investigation in Figure 2 – (*) means that values vary and determine the Pr(Available Zone) on x-axis.

Parameters	Direction	FD	SD	CE
N		100	100	100
$\mu_{switch-A-U}$ (*)		30	30	30
$\mu_{switch-U-A}$ (*)		30	30	30
μ_{wait}		1	1	1
μ_{reach}	F	9	9	9
	B	1	1	1
$\mu_{goStraight}$	F	9	9	9
	B	1	1	1
μ_{turn}	F	9	9	9
	B	1	1	1
$\mu_{goAround}$	F	27	27	27
	B	3	3	3
μ_{follow}	F	–	1	–
	B	–	1	–
μ_{fail}	F	–	–	1
	B	–	–	1
μ_{ask}	F	–	–	1
	B	–	–	1
$\mu_{refresh}$	F	–	–	300
	B	–	–	300

ware architects can tune these parameters and get quantitative information. For instance, the performance results obtained in our preliminary investigation (see Figure 2) are derived from numerical values of model parameters shown in Table III.

1) Fully-Decentralized: Figure 5 depicts N robots that move forward and backward between the initial and the target zones, separated by a temporary obstacle, such as a door. When this pattern is adopted, robots never communicate with each other, they take decisions based on their observations. First, they *wait* to receive a task. In the forward block, robots *reach* the obstacle and *go straight* if they can overcome it (e.g., the door is open), otherwise they must *turn* and *go around* the

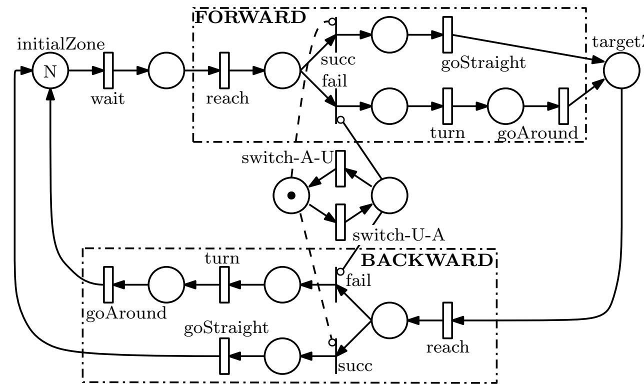


Fig. 5: Fully-decentralized architecture.

obstacle. After reaching the target zone, robots must go back, as modelled by the backward block. We assume that robots traveling in opposite directions do not hamper each other. As shown in Table III, for this architectural pattern, see fully-decentralized (FD) column, software architects can set the following parameters: N, i.e., the number of robots; $\mu_{switch-A-U}$ and $\mu_{switch-U-A}$ represent the average time regulating the (un)availability of zones. Please note that in Table III these two parameters are distinguished from the others with this symbol (*), since they contribute to establishing the probability of a zone that is then shown on the x-axis of Figure 2. As said in Section III-A, we recall that the probability that a zone is available is computed as $\mu_{switch-A-U}/(\mu_{switch-A-U}+\mu_{switch-U-A})$. This means that the values in Table III (i.e., 30-30) lead to determine $Pr(AvailableZone) = 0.5$. Further parameters are: μ_{wait}, i.e., the average time for a robot to receive a task; μ_{reach}, i.e., the average time for reaching the obstacle that can be different in forward (F) and backward (B) paths; similarly, $\mu_{goStraight}$, μ_{turn}, and $\mu_{goAround}$ are the average times used for the corresponding timed transitions (see Figure 5).

2) Semi-Decentralized: Figure 6 shows the performance model of the SD pattern by which robots communicate with their peers if a certain zone is unavailable. Places and transitions with bold names have been added to explicitly model the architectural aspects. When a robot fails to go straight because of an obstacle, it sends a *notice* to all the other robots in the same room. These latter robots *follow* such a recommendation of not proceeding and *go around* the obstacle without approaching it. The robot elected as the spreader of such information still needs to *turn* and *go around* the obstacle. As reported in Tables II and III, this pattern additionally requires the setting of μ_{follow}, i.e., the average time spent to communicate with the robot spreading the notice.

3) Centralized: Figure 7 (similarly to Figure 6, bold names explicitly target the architectural aspects) depicts the centralized architecture by which robots interact with a central coordinator in case of an impediment. When a robot fails to go straight because of an obstacle, it sends a *notice* to the coordinator that is in charge of dispatching such information to all other robots. Differently from the semi-decentralized architecture, the information is sent not only to the robots

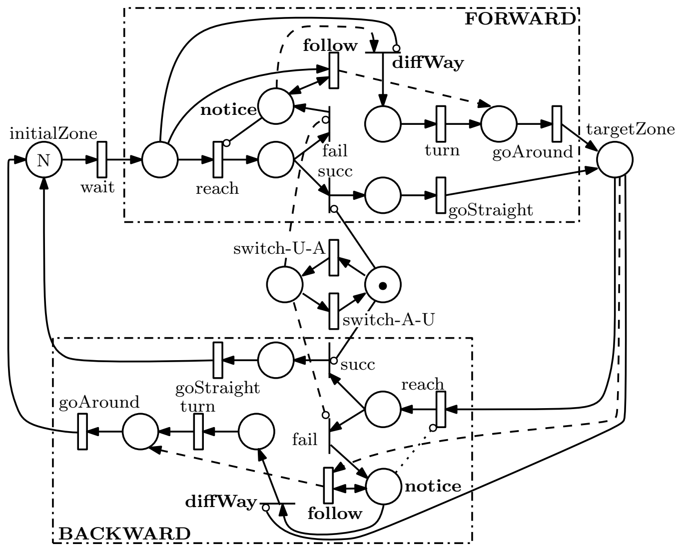


Fig. 6: Semi-decentralized architecture.

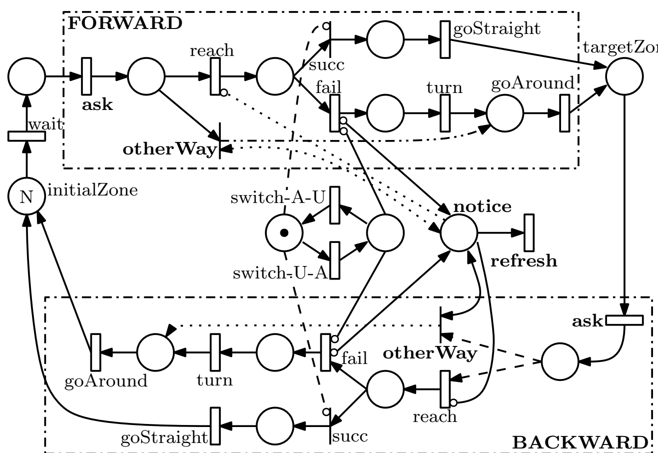


Fig. 7: Centralized architecture.

occupying the same room of the spreader, but to any robot approaching the obstacle. The coordinator periodically can *refresh* the information on the (un)availability of zones, waiting for a robot acknowledging the presence of an obstacle. As shown in Tables II and III, this pattern requires the setting of: μ_{fail}, i.e., the average time required for the communication between the robot that notifies the presence of an obstacle to the central coordinator; μ_{ask}, i.e., the average time required for the robots to ask information about the status of a zone to the central coordinator; $\mu_{refresh}$, i.e., the average time for triggering the control on the (un)availability of zones.

IV. Experimental Evaluation

In this section, we describe the results obtained by analyzing scenarios that might be of interest to software architects. Note that our prediction performance results are not compared with actual measurements from the system implementation, since this is out of this paper scope. Our assumption is that the prediction methods are sound and provide accurate prediction results, as assessed in other works in the literature [25], [26].

A. Research questions

The purpose of our experimental evaluation is twofold: *i)* it shows our method can be applied to real medium-sized scenarios; *ii)* it provides empirical evidence of the impact of different architectural patterns on the performance characteristics of a CPS. In particular, we aim to answer two research questions:

RQ1: What are the performance characteristics of architectural patterns when applied to robots moving through multiple temporary (un)available zones? Which system parameters affect the optimality of patterns?

RQ2: What are the performance characteristics of architectural patterns in case of permanent (un)available zones? What happens when changing the percentage of different types of robots in the system?

To answer these questions, we built performance models representing two different scenarios, namely S_1 and S_2 in the following. Performance models are simulated using JSIMgraph [27] installed on a cluster hosting a virtual machine with 16 vCPU and 32GB memory. In the worst case, each simulation takes 10 minutes, that is acceptable recalling that the analysis is performed at design time, see Section III.

B. Scenario S_1: Multiple Temporary Obstacles

Description. Robots move from an initial to a target zone to deliver medicines, as shown in Figure 8. Two routes connect the zones: *1)* a short route with a finite number of *temporary* obstacles, i.e., three doors, and *2)* a long route. Robots can choose one of the two routes when they arrive at a fork. Those that choose the short way may find themselves unable to reach their destination due to a closed door, and they need to go back to the fork. The later a robot is blocked by a closed door, the longer it takes to go back to the fork and follow the alternative route. Once medicines are delivered, robots must go back to the initial zone before being able to convey other items. Robots must choose again one of the two routes (i.e., short or long) to reach their destination. Doors may show a different probability to be closed or open. In fact, each door is modelled independently of others, i.e., the status of a door does not depend on others.

Parametrization. Table IV reports the numerical values (timings are expressed in seconds) of model parameters used in Scenario S_1 that considers three temporary obstacles, i.e., doors. For all parameters (already discussed in Section III), we provide values for: (i) directions, i.e., forward (F) and backward (B); (ii) door instances, namely D_1, D_2, and D_3; (iii) architectural patterns (i.e., FD, SD, and CE). For the sake of simplicity, all parameters (except μ_{turn}) show no difference when considering different directions or doors across the considered patterns. This choice is to enable a fair comparison among the different architectural patterns. Differences observed for μ_{turn} are due to the distance of each door from the fork, e.g., a robot moving from the initial to the target zone (i.e., forward) needs 5, 15, or 25 seconds to go back to

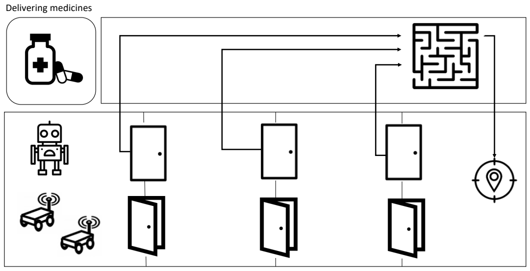


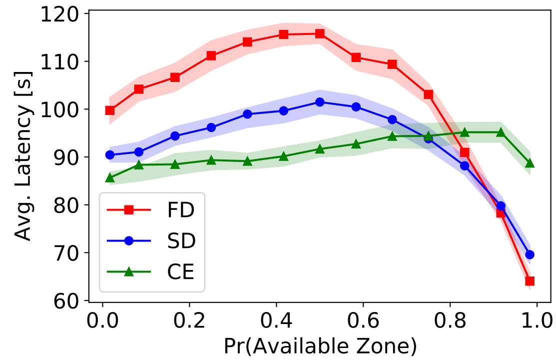
Fig. 8: Scenario S_1: multiple temporary (un)available zones along the path to reach the target destination.

TABLE IV: Numerical values of model parameters for S_1.

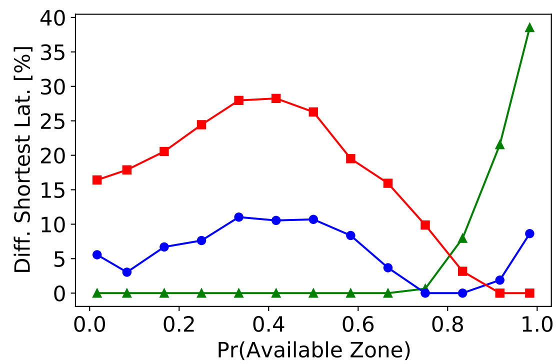
Parameters	Direction	Door	FD	SD	CE
N			100	100	100
$\mu_{switch-A-U}+$ $\mu_{switch-U-A}$			60	60	60
μ_{wait}			10	10	10
μ_{reach}	F / B	D^*	5	5	5
$\mu_{goStraight}$	F / B	D^*	5	5	5
μ_{turn}	F	D_1	5	5	5
		D_2	15	15	15
		D_3	25	25	25
	B	D_1	25	25	25
		D_2	15	15	15
		D_3	5	5	5
$\mu_{goAround}$	F / B	D^*	40	40	40
μ_{follow}	F / B	D^*	–	1	–
μ_{fail}	F / B	D^*	–	–	1
μ_{ask}	F / B	D^*	–	–	1
$\mu_{refresh}$	F / B	D^*	–	–	60

the fork if door D_1, D_2, or D_3, respectively, blocks its way. When the robot moves from the target to the initial zone (i.e., backward), it approaches doors in the opposite order (i.e., first D_3, then D_2, and finally D_1) and this is why it needs 5, 15, or 25 seconds, respectively, to go back to the fork.

Results. Figure 9 shows the results obtained by simulating Scenario S_1 with parameters given in Table IV. Solid lines in Figure 9a depict the average system response time (i.e., the time spent by a robot for going from the initial to the target zone, then back to the initial zone) against the probability that *each single* door is open. For instance, the 0.5 value on the x-axis of Figure 9a means that D_1, D_2, and D_3 are open with a probability of 0.5, whereas the overall system probability (all doors open at the same time) is given by their product, i.e., 0.125. Results are collected with 95% confidence intervals shown in Figure 9a by the shaded areas. Figure 9b depicts how far is each architectural pattern (in terms of average latency) from the one allowing robots moving with the shortest latency, i.e., 0% on the y-axis represents the optimal architectural pattern. In case of low probability for each door to be open, i.e., small values of $Pr(AvailableZone)$, the FD architectural pattern shows the worst performance since robots do not synchronize or exchange information. This architectural pattern minimizes the system response time when the probability that each door is open is larger than 0.9 since there is no communication overhead. SD and CE



(a) Average Latency

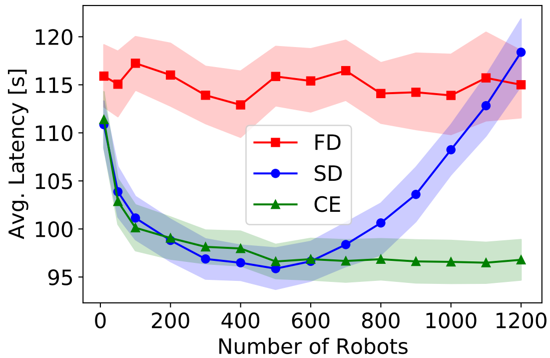


(b) Distance from the shortest latency

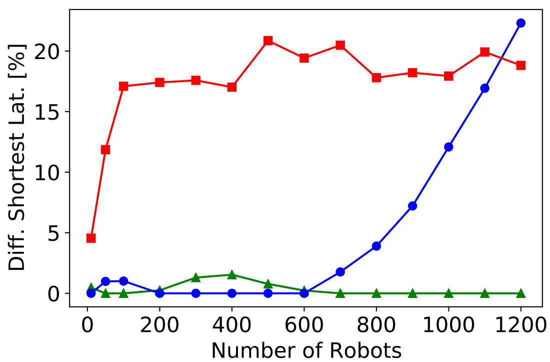
Fig. 9: System performance of Scenario S_1 plotted against the probability that each door is open. Numerical values of model parameters are given in Table IV.

architectural patterns show similar performance when the probability that each door is open is between 0% and 75%. Benefits of using the SD architectural pattern are maximized for $0.75 \leq$ Pr(Available Zone) ≤ 0.9. In this case, robots finding a closed door minimize the communication overhead by exchanging messages only with other robots approaching the same door. The CE architectural pattern shows minimum latency for $0 \leq$ Pr(Available Zone) < 0.75 since all robots are aware of door status thanks to the central coordinator.

Figure 10 reports the performance of each architectural pattern against the number of robots in the system when setting: $Pr(AvailableZone) = 0.5$ and $\mu_{refresh} = 10$ seconds. These experiments show that the proposed approach is highly scalable since it is able to handle at least 1.2k robots. Large fleets are generally made of hundreds of robots [17]. It is worth noting that the efficiency of the CE architectural pattern increases with the number of robots. When there are less than 600 robots in the system, the CE and SD architectural patterns show equivalent performance. The CE architecture keeps performing well also with a high number of robots thanks to the short communication times (i.e., μ_{fail} and μ_{ask}, see Table IV). A fragility of CE might be represented by the central node that becomes overwhelmed of requests. Parameters of our model are set to avoid such a scenario, but we plan to further investigate this in our future research. The



(a) Average Latency



(b) Distance from the shortest latency

Fig. 10: System performance of Scenario S_1 plotted against the number of available robots when Pr(Available Zone) = 0.5. All model parameters, except N (varying, see x-axis) and $\mu_{refresh}$ (10 seconds), are the same as shown in Table IV.

average response time of the SD architectural pattern shows a convex behavior, indeed high latency values are observed with both a few robots in the system (i.e., when it is difficult to spread the information about the door status) and many robots (i.e., when communicating with other robots in the same zone is expensive). The SD architecture shows the worst performance in case of more than 1.1k robots, and this is due to the overhead accumulated by the increasing number of robots spreading the information on the status of the three doors.

> **RQ1:** In summary, there is not a unique architectural pattern that always overcomes the others in case of temporary (un)available zones. The optimal pattern depends on the probability that a robot finds its way blocked, the number of robots, and the communication overhead.

C. Scenario S_2: Temporary and Permanent Obstacles

Description. In this scenario, initial and target zones are located on two different floors that are connected by the stairs and an elevator, see Figure 11. The stairs may be *momentarily* inaccessible (e.g., being overcrowded or due to obstacles), while the elevator has a *permanent* finite capacity, i.e., it can move only a subset of robots at the same time. Two types of robots are considered in this scenario: *1)* two-legged robots can reach the other floor using either the stairs or the elevator; *2)* wheeled robots can move to the other floor only using the elevator. We assume that both types of robots move at the same speed. Two-legged robots first try to change the floor by taking the stairs and, if there are obstacles blocking the way, they go back to the elevator. Even if the elevator may allow robots to reach their destination faster than the stairs, its finite capacity makes robots spend time in line waiting for their turn to use the elevator. This can dramatically extend the time required to reach the other floor by using the elevator. The waiting time is further extended since we intentionally consider there is only one elevator serving both floors.

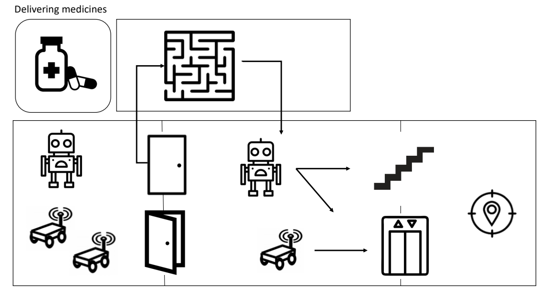
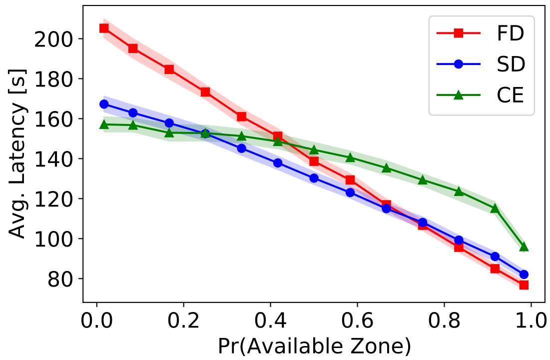


Fig. 11: Scenario S_2: multiple temporary and permanent (un)available zones along the path.

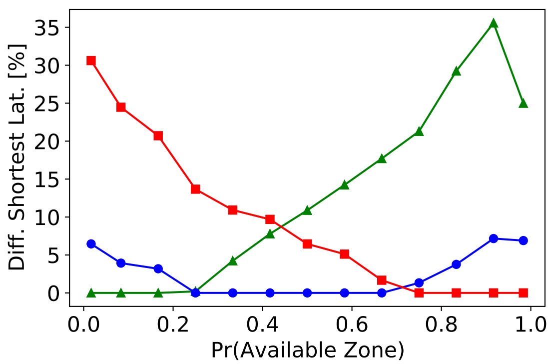
TABLE V: Numerical values of model parameters for S_2.

Parameters	Direction	Zone	FD	SD	CE
$N_{two-legged}$			80	80	80
$N_{wheeled}$			20	20	20
$\mu_{switch-A-U}+$ $\mu_{switch-U-A}$			60	60	60
μ_{wait}			10	10	10
μ_{reach}	F / B	Door	5	5	5
		Stairs	15	15	15
$\mu_{goStraight}$	F / B	Door	5	5	5
		Stairs	15	15	15
μ_{turn}	F / B	Door	5	5	5
		Stairs	15	15	15
$\mu_{goAround}$	F / B	Door	40	40	40
		Stairs	–	–	–
K	F / B	Door	–	–	–
		Stairs	20	20	20
μ_{moving}	F / B	Door	–	–	–
		Stairs	15	15	15
μ_{empty}	F / B	Door	–	–	–
		Stairs	15	15	15
μ_{follow}	F / B	Door	–	1	–
		Stairs	–	1	–
μ_{fail}	F / B	Door	–	–	1
		Stairs	–	–	1
μ_{ask}	F / B	Door	–	–	1
		Stairs	–	–	1
$\mu_{refresh}$	F / B	Door	–	–	60
		Stairs	–	–	60

Parametrization. Table V shows the numerical values of model parameters used in Scenario S_2 that considers initial and target zones on two different floors. Similarly to the previous

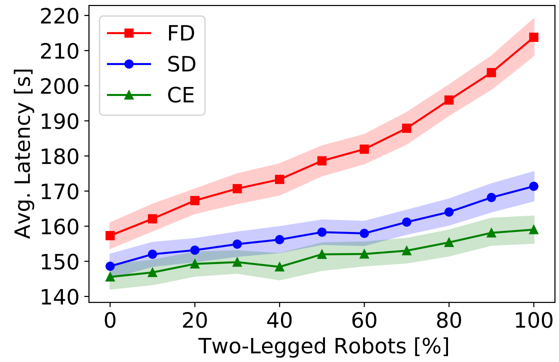


(a) Average Latency

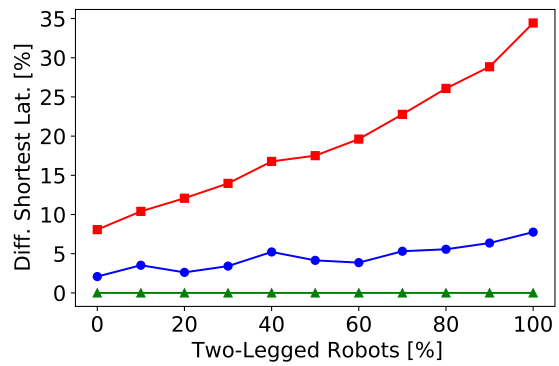


(b) Distance from the shortest latency

Fig. 12: System performance of Scenario S_2 plotted against the probability that each door is open and the stairs are accessible. Numerical values of model parameters are given in Table V.



(a) Average Latency



(b) Distance from the shortest latency

Fig. 13: System performance of Scenario S_2 plotted against percentage of two-legged robots when Pr(Available Zone) = 0.083. All model parameters, except $N_{two-legged}$ and $N_{wheeled}$, are the same as shown in Table V.

scenario, we report values for each direction (i.e., forward and backward), zone (i.e., Door and Stairs), and the considered architectural patterns (i.e., FD, SD, and CE). Here we assume that there might be different costs to interact with diverse dynamic spaces, e.g., $\mu_{reach} = 5$ for a door and 15 for stairs.

Results. Figure 12 depicts the performance of the three architectural patterns applied to Scenario S_2 against the probability that some obstacles (i.e., a closed door or over-crowded stairs) make robots take alternative (i.e., longer) routes. Figure 12a shows the average system response time. Figure 12b depicts how far is the response time of each architectural pattern from the shortest one observed during the experiment. In this scenario, the CE architectural pattern shows good performance only when the probability of robots finding a blocked way is high. Note that a long refresh time (i.e., $\mu_{refresh}$) further worsen the observed latency by decreasing the frequency with which the central coordinator updates stored information about the obstacle (i.e., door and stairs) status. Results for longer (i.e., $\mu_{refresh}$) are not shown here for the sake of space. The SD architectural pattern performs better than CE for $0.3 \leq$ Pr(Available Zone) ≤ 0.7. The motivation for this behavior is that SD includes communication among peers occupying the same zone, whereas CE keeps propagating information that is useless for robots that are far from the obstacle. As previously observed for Scenario S_1, the FD architectural pattern minimizes the latency when the probability that there are no obstacles is high. In this case, reducing the number of exchanged messages is the winning strategy.

Figure 13 depicts the performance of architectural patterns against the ratio of two-legged (and wheeled) robots. These results are obtained by setting Pr(Available Zone) $= 0.083$, i.e., $5/60 = \mu_{switch-A-U}/(\mu_{switch-U-A} + \mu_{switch-A-U})$. Such a low value is selected for modelling many robots using the elevator to move among floors. The CE architectural pattern always provides the shortest latency. Benefits of using this pattern instead of others increase with a higher percentage of two-legged robots in the system. This is due to the key role of the central coordinator that alerts two-legged robots of not taking the stairs (recall that probability of being available is intentionally set low) and using the elevator.

RQ2: In summary, always-optimal architectural patterns do not exist also for permanent (un)available zones. Having more two-legged robots does not affect the optimal architectural pattern but worsen the system performance.

D. Threats to Validity

We are aware that generalization of results (i.e., *external validity*) is not guaranteed, since our models have been applied to two medium-sized scenarios only, and we plan to develop larger scenarios as future work. However, at the current stage our focus is to raise the attention of software architects pointing out that design alternative patterns show very different performance characteristics when subject to space dynamics.

To mitigate threats to *internal validity*, we designed our experiments with the goal of having a direct manipulation on the performance indices of interest. For instance, the three architectural patterns share parameter values to avoid misleading effects that cannot be traced back to root causes. Setting numerical values to input parameters is indeed an open issue in the software performance engineering domain [28]. To improve this point, the probabilities of available zones might be derived with the introduction of a monitor that collects data for a certain time frame and produce some statistics. Besides, other parameters can be further detailed, e.g., different robots may show diverse performance characteristics (e.g., the speed of going straight). This implies the adoption of coloured Petri Nets [29] that may represent an extension of our current modelling. Moreover, the choice of using GSPN as the target notation for modelling the performance does not reduce the applicability of our method. As future work, we plan to experiment with further notations (e.g., queueing Petri nets [30], [31]) to investigate their usability and scalability.

To smooth *construct validity* threats (i.e., the assessment of the validity of the results used during our experimentation) and assess *statistical validity* of collected performance indices, we set that all simulations undergo a 95% confidence interval to monitor the accuracy of numerical results.

V. Related Work

The work presented in this paper relates to two main streams of research that we review in the following.

Architecting Cyber-Physical Systems. A preliminary study in this direction is provided in [32], where authors discuss the open challenges; dealing with the performance characteristics of CPS is identified as a relevant matter. An evolution of this study is presented in [33], performance aspects are even more detailed and there are some further goals that emerge as of key relevance, i.e., timeliness and dynamic path planning that are both considered by our method. The continuous monitoring of environmental conditions is proposed in [8], and an architectural description is proposed for modelling cyber-physical spaces. Dynamic constraints of CPS architectures are investigated also in [34], and an industrial case study on autonomous transportation robots is used for defining a variability modelling approach in charge of documenting such constraints. Architecture-based self-adaptation is tackled for CPS, please refer to the systematic literature review in [9], but also for IoT several approaches recently emerged [11], [35], [36]. Our work makes use of the architectural patterns for the adaptation of CPS [16], and we are interested in exploiting their performance-related characteristics.

Performance modelling of Dynamic Spaces. The modelling of evolving cyber-physical spaces has been proposed in [13] for verification purposes, however this approach relies on a logic-based specification of system properties that are later analyzed with probabilistic model checking. The efficiency of this verification engine has been recently improved in [37] where a slicing technique is introduced to transform the specification of a model into equivalent sub-models that achieve better scalability since they are tailored for analyzing specific requirements. The analysis of spatio-temporal properties of stochastic systems recently gained the attention of several researchers. For instance, in [38] a spatio-temporal reach and escape logic, namely STREL, is introduced to verify spatial operators, later refined in [39] to keep track of the evolution of the satisfaction of system properties. More recently, in [40] a tool has been developed to monitor spatio-temporal properties of CPS, where space is modelled as a weighted graph whose quantities can change overtime.

Summarizing, our work mainly differentiates from the state-of-the-art since we explicitly target the performance characteristics of architecting CPS, and our method shows the goal to support software architects in the task of evaluating multiple design patterns and selecting the optimal one depending on the considered scenario and its physical space dynamics.

VI. Conclusion

In this paper, we present a novel approach to model and analyze the performance characteristics of different architectural patterns in the context of cyber-physical dynamic spaces. We propose a set of performance models to investigate the impact of architectural patterns on system performance, and results confirm the usefulness of our models as support to software architects in the task of evaluating different design alternatives. Our experimental evaluation focuses on assessing the system performance of different architectural patterns, and it reinforces our initial guess that it is indeed helpful to analyze the circumstances triggered by the dynamic spaces, and to switch among such patterns accordingly. The main contribution of our method is to provide quantitative information to raise the attention of software architects to the performance evaluation of different design alternatives.

In future work, we plan to address all the limitations discussed as part of threats to validity. Besides, we are aware that building performance models requires a high level of expertise and a deep understanding of the CPS. To support software architects in re-applying our method in another problem context, we plan to build a framework that semi-automatically generates performance models and produces results, on the basis of some configuration settings. Moreover, we are interested to apply the approach in different industrial domains to investigate its usefulness across diverse applications and more complex scenarios, e.g., robots that hamper each other when traveling in opposite directions due to small hallways. This way, we plan to further investigate the scalability of our method with more complex systems in different domains.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work has been partially funded by MIUR PRIN project 2017TWRCNB SEDUCE (Designing Spatially Distributed Cyber-Physical Systems under Uncertainty). We are also grateful to the Computing and Network Service for their support in our experiments on the U-LITE cluster at INFN, LNGS, L'Aquila, Italy.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[2] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, "Design, modelling, simulation and integration of cyber physical systems: Methods and applications," *Computers in Industry*, vol. 82, pp. 273–289, 2016.

[3] M. Shaw, D. Garlan *et al.*, *Software architecture*. Prentice Hall Englewood Cliffs, 1996, vol. 101.

[4] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*. Addison-Wesley Professional, 2000.

[5] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[6] C. Gerking and D. Schubert, "Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures," in *Proceedings of the International Conference on Software Architecture (ICSA)*, 2019, pp. 61–70.

[7] B. Tenbergen, M. Daun, P. A. Obe, and J. Brings, "View-Centric Context Modeling to Foster the Engineering of Cyber-Physical System Networks," in *Proceedings of the International Conference on Software Architecture (ICSA)*, 2018, pp. 206–216.

[8] H. Muccini and M. Sharaf, "CAPS: Architecture Description of Situational Aware Cyber Physical Systems," in *Proceedings of the International Conference on Software Architecture (ICSA)*, 2017, pp. 211–220.

[9] H. Muccini, M. Sharaf, and D. Weyns, "Self-adaptation for cyber-physical systems: a systematic literature review," in *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2016, pp. 75–81.

[10] M. Sharaf, M. Abughazala, H. Muccini, and M. Abusair, "An Architecture Framework for Modelling and Simulation of Situational-Aware Cyber-Physical Systems," in *Proceedings of the European Conference on Software Architecture (ECSA)*, 2017, pp. 95–111.

[11] J. Cámara, H. Muccini, and K. Vaidhyanathan, "Quantitative Verification-Aided Machine Learning: A Tandem Approach for Architecting Self-Adaptive IoT Systems," in *Proceedings of the International Conference on Software Architecture (ICSA)*, 2020, pp. 11–22.

[12] C. Smith and L. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[13] C. Tsigkanos, T. Kehrer, and C. Ghezzi, "Modeling and verification of evolving cyber-physical spaces," in *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2017, pp. 38–48.

[14] V. Ciancia, S. Gilmore, G. Grilletti, D. Latella, M. Loreti, and M. Massink, "Spatio-temporal model checking of vehicular movement in public transport systems," *International Journal on Software Tools for Technology Transfer*, vol. 20, no. 3, pp. 289–311, 2018.

[15] A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa *et al.*, "Modelling and analysing resilient cyber-physical systems," in *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2019, pp. 70–76.

[16] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf, "Patterns for self-adaptation in cyber-physical systems," in *Multi-disciplinary engineering for cyber-physical production systems*. Springer, 2017, pp. 331–368.

[17] F. Weidinger, N. Boysen, and D. Briskorn, "Storage assignment with rack-moving mobile robots in KIVA warehouses," *Transportation Science*, vol. 52, no. 6, pp. 1479–1495, 2018.

[18] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," vol. 4, no. 26, pp. 1–13.

[19] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance evaluation*, vol. 67, no. 8, pp. 634–658, 2010.

[20] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[21] R. David and H. Alla, "Petri nets for modeling of dynamic systems: A survey," *Automatica*, vol. 30, no. 2, pp. 175–202, 1994.

[22] P. Yuan, K. Zheng, X. Xiong, K. Zhang, and L. Lei, "Performance modeling and analysis of a Hyperledger-based system using GSPN," *Computer Communications*, vol. 153, pp. 117–124, 2020.

[23] D. Carvalho, L. Rodrigues, P. T. Endo, S. Kosta, and F. A. Silva, "Mobile Edge Computing Performance Evaluation using Stochastic Petri Nets," in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–6.

[24] H. Hu, J. Yu, Z. Li, J. Chen, and H. Hu, "Modeling and Analysis of Cyber–Physical System Based on Object-Oriented Generalized Stochastic Petri Net," *IEEE Transactions on Reliability*, pp. 1–15, 2020.

[25] A. Rogge-Solti and M. Weske, "Prediction of business process durations using non-Markovian stochastic Petri nets," *Information Systems*, vol. 54, pp. 1–14, 2015.

[26] G. Balbo and G. Ciardo, "On Petri Nets in Performance and Reliability Evaluation of Discrete Event Dynamic Systems," in *Carl Adam Petri: Ideas, Personality, Impact*. Springer, 2019, pp. 173–185.

[27] M. Bertoli, G. Casale, and G. Serazzi, "JMT: performance engineering tools for system modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, 2009.

[28] A. B. Bondi, *Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice*. Pearson Education, 2015.

[29] K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer Science & Business Media, 2013, vol. 1.

[30] S. Kounev, "Performance modeling and evaluation of distributed component-based systems using queueing petri nets," *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 486–502, 2006.

[31] M. Frank, A. Hakamian, and S. Becker, "Defining a Formal Semantic for Parallel Patterns in the Palladio Component Model Using Hierarchical Queuing Petri Nets," in *Proceedings of the European Conference on Software Architecture (ECSA)*, 2020, pp. 381–394.

[32] I. Malavolta, H. Muccini, and M. Sharaf, "A preliminary study on architecting cyber-physical systems," in *Proceedings of the European Conference on Software Architecture (ECSA), Companion Volume*, 2015, pp. 1–6.

[33] I. Crnkovic, I. Malavolta, H. Muccini, and M. Sharaf, "On the use of component-based principles and practices for architecting cyber-physical systems," in *Proceedings of the International Symposium on Component-Based Software Engineering (CBSE)*, 2016, pp. 23–32.

[34] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, and J. S. Zernickel, "Model-based documentation of dynamicity constraints for collaborative cyber-physical system architectures: Findings from an industrial case study," *Journal of Systems Architecture*, vol. 97, pp. 153–167, 2019.

[35] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying Architecture-Based Adaptation to Automate the Management of Internet-of-Things," in *Proceedings of the European Conference on Software Architecture (ECSA)*, 2018, pp. 49–67.

[36] M. Saelens, Y. Kinoo, and D. Weyns, "HeyCitI: Healthy Cycling in a City using Self-Adaptive Internet-of-Things," in *Proceedings of the International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), Companion Volume*, 2020, pp. 226–227.

[37] C. Tsigkanos, N. Li, Z. Jin, Z. Hu, and C. Ghezzi, "Scalable Multiple-View Analysis of Reactive Systems via Bidirectional Model Transformations," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2020, pp. 993–1003.

[38] E. Bartocci, L. Bortolussi, M. Loreti, and L. Nenzi, "Monitoring mobile and spatially distributed cyber-physical systems," in *Proceedings of the International Conference on Formal Methods and Models for System Design, (MEMOCODE)*, 2017, pp. 146–155.

[39] L. L. Vissat, M. Loreti, L. Nenzi, J. Hillston, and G. Marion, "Analysis of Spatio-temporal Properties of Stochastic Systems Using TSTL," *ACM Trans. Model. Comput. Simul.*, vol. 29, no. 4, pp. 20:1–20:24, 2019.

[40] E. Bartocci, L. Bortolussi, M. Loreti, L. Nenzi, and S. Silvetti, "MoonLight: A Lightweight Tool for Monitoring Spatio-Temporal Properties," in *Proceedings of the International Conference on Runtime Verification (RV)*, 2020, pp. 417–428.