



HAL
open science

Towards Leveraging the Concept of Influence to Enhance Collaborative Cyber-Physical Systems Development

Bárbara da Silva Oliveira, Nicolas Ferry, Julien Deantoni

► To cite this version:

Bárbara da Silva Oliveira, Nicolas Ferry, Julien Deantoni. Towards Leveraging the Concept of Influence to Enhance Collaborative Cyber-Physical Systems Development. MPM4CPS 2024 - Multi-Paradigm Modeling for Cyber-Physical Systems @MODELS 2024, Sep 2024, Linz, Austria. 10.1145/3652620.3688568 . hal-04695644

HAL Id: hal-04695644

<https://inria.hal.science/hal-04695644v1>

Submitted on 12 Sep 2024


HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.


L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Towards Leveraging the Concept of Influence to Enhance Collaborative Cyber-Physical Systems Development

Bárbara da Silva Oliveira 
barbara.da-silva-oliveira@etu.univ-
cotedazur.fr
Université Côte d’Azur,
I3S/INRIA Kairos
Sophia Antipolis, France

Nicolas Ferry 
nicolas.ferry@univ-cotedazur.fr
Université Côte d’Azur,
I3S/INRIA Kairos
Sophia Antipolis, France

Julien Deantoni 
julien.deantoni@univ-cotedazur.fr
Université Côte d’Azur,
I3S/INRIA Kairos
Sophia Antipolis, France

Abstract

Cyber-Physical Systems development requires the collaboration of various stakeholders from multiple domains. Whilst the functional exchanges between the design artifacts are captured using classical software engineering methods, it is common to have interactions between different domains that go beyond mere functional ones. These may not even be fully understood or known at design-time, making more difficult the rationalization of the design choices and generally speaking the collaboration. Collaborative simulation appears as a solution to observe the emerging behaviors of such systems. However, it does not help identifying the cause of changes in the emerging behaviors, e.g., when different teams made changes concurrently on their artifacts. In this paper, we propose to make explicit how some elements of the design artifacts are known (more or less precisely) to *influence* the behavior of other design artifacts. We argue this is beneficial for the collaborative development of CPS since it provides an enrichment of the (potentially not fully understood) semantics that exist between different elements of the systems, beyond mere functional ones. Based on our example, the paper presents a first categorization for our notion of *influences* as well as a discussion about potential usages and their benefits.

Keywords

Cyber-Physical Systems, Collaborative System Design, Collaborative Simulation, System Behavior

1 Introduction

Cyber-Physical Systems (CPS) have emerged as a transformative technology, integrating physical and computational processes. CPS are composed of interconnected parts, each designed to fulfill specific roles within the system [3, 13]. These parts are typically defined based on domains of expertise, such as hardware, software, network, and can be further divided into more specialized parts. They are often developed independently in silos by different developers, teams, and organizations.

Before delivery, the material created during this process (e.g., code, models, configurations), hereafter called design artifact [6], needs to be evaluated realizing simulations, experimentation, and tests. However, analysing the behavior of individual parts running isolated simulations is not sufficient as interaction across design artifacts from the different parts needs to be considered.

Collaborative simulation has emerged as an essential technique for evaluating the interactions between multiple system parts. Simulation results related to a part of the system are considered in relation to others, providing simulation outcomes that reflect the

emerging behavior of the system [5]. However, despite paving the way of understanding the complete system behavior, some limitations still remain.

Co-simulation typically produces large volume of raw logs, from which it is difficult to assess the impact of a novel design choice on the whole system. Design choices made in one design artifact may have far-reaching consequences, especially when dealing with large and heterogeneous design spaces as in CPS [12]. Design choices in one part of the system may influence other parts’ behavior, possibly violating their requirements or worsening their outcomes. Without clear visibility on how different artifacts influence each other, beyond mere functional exchanges captured using classical software engineering methods, it becomes difficult to identify (statically or dynamically using co-simulation) the consequences of changes and to ensure that the system behaves as intended. As a result, it becomes challenging to make accurate decisions and progress through the development lifecycle.

In this paper we argue that, beyond the abstractions, and specifications identified by a development team to reach the system’s desired goals, making explicit how design choices on design artifacts, or elements of design artifact, *influence* the behavior of other design artifacts is beneficial for the collaborative development of CPS.

The objective of this paper is to propose a definition of the concept of influence together with a conceptual workflow to exploit this novel knowledge during CPS development. Our contribution is focused on the definition and classification of influences, which we illustrate and motivate this concept using a mobile robot use case. We describe a conceptual workflow for the development of CPS, incorporating the concept and the exploration of influences into the system development. By improving the comprehension of these influences, we intend to enhance decision-making and collaboration between the various users involved in a system development.

The remainder of the paper is organized as follows. Section 2 presents our motivating example. Section 3 introduces the concept of influences. Section 4 outlines the conceptual framework we aim to implement. Section 5 positions our contribution within the state of the art and Section 6 concludes.

2 Mobile Robot Use Case

In this section, we introduce our exploration robot as a use case from which we extracted design scenarios that motivate the need for extra information to ease collaborative development.

The robot used is a Cherokee 4WD Arduino mobile robot [4]. This simple robot, equipped with useful but limited hardware, is

often used for student competitions/teaching and research activities. The requirements for the robot are simple: the robot must explore an indoor environment while avoiding obstacles.

The robot moves by using two gearboxed DC motors. It uses a RoMeo V1 Board, which is based on Arduino Uno and includes an ATmega328 microcontroller. The robot is equipped with a servo rotary motor that allows an ultrasonic sensor to sweep the environment and detect obstacles. The ultrasonic sensor measures the proximity of objects with the transmission of ultrasonic pulses. It is worth mentioning that our contribution is not focused on the domain of robotics but on the exploration of the system development process for motivating our work. In the following, we detail the different parts of the system, their model, and key design artifacts.

2.1 System Design Model Description

The system is divided into the following six parts (see Figure 1):

- Wheel Controller** This part receives move commands and applies the power to the right and left wheels accordingly. Examples of move commands are forward, stop, or turnLeft.
- Dynamic Detection** : Checks for dangerous obstacles while sweeping the environment in front of the robot. This part is active while the robot is moving.
- Static Detection** : scans for obstacles in front and on the sides of the robot; for a better understanding of the environment and accurate avoidance decisions.
- Avoidance Strategy** : From an ordered set of detected obstacles, decides the maneuvers necessary to avoid detected obstacles. These maneuvers are provided as timed moves, i.e., a set of move command and their duration.
- Normal Move** : Sends move commands to explore the environment.
- Avoidance Move** : Receives a set of timed moves and applies them iteratively for the specified duration.

These different parts are not all enabled at the same time and are orchestrated as specified by the State Chart represented in Figure 2. The name of the states corresponds to the name of the parts introduced in Figure 1. When a state is entered, the behavior of the part is started. When a state is exited, the behavior is aborted. At the start of the system, both the normal move and the dynamic detection (of obstacles) are performed. If an obstacle is detected, both stop, and the static detection (of obstacles) is started. When finished, the avoidance strategy is executed. Either the avoidance strategy decides to continue as is and reenter the parallel state with the normal move and dynamic detection; or some maneuvers to avoid the obstacles are computed; followed by the parallel execution of the avoidance move and the dynamic detection.

Finally, the behavior of some of the individual system parts is detailed in Figure 3 using activity diagrams. These activity diagrams are given for record and we detail only specific parts of each behavior here.

In the Dynamic Detection, the servo motor position is changed (to sweep the environment); followed by an obstacle check by using the ultrasonic sensor. The designer of this part's behavior decided to consider that a dangerous obstacle is one whose distance is less than 15 cm from the robot (see decision node's predicate). If it occurs, an event is emitted, firing a transition in the orchestrator

state chart. If not, sleep is realized before restarting the obstacle checking.

During Static Detection, the stop move command is sent in order to realize a precise object scanning. The sweep of the environment is done almost like in the dynamic detection part except that the sweep is wider, all detected obstacles are put in a FIFO, and that no sleeps are realized. Once the whole sweep of the environment is done, the behavior is stopped, and the avoidance strategy is started.

In the avoidance strategy, if the danger is confirmed, a set of timed move commands are generated and sent to the avoidance move, which is started in turn (in parallel with the dynamic detection). Avoidance commands are then sent as appropriate. Once all maneuvers are completed with no new obstacle detection, the normal move is restarted.

2.2 Motivating scenario

This use case is small but sufficient to illustrate some difficulties that may appear during the collaborative development of a CPS. We illustrate these difficulties through the definition of scenarios.

Alice, Bob, and Charlie are software developers working on the autonomous robot. Alice works on the Static and Dynamic Detection parts, Bob works on the Avoidance Strategy part, and Charlie works on the Wheel Controller part. They collaborate through the usage of a common artifact repository and co-simulation of the developed artifacts, the physical environment, and the model of the robot.

At some point in the development, Charlie was required to speed up the exploration to better handle large buildings. He decided to increase the power on each wheel of the robot with the objective of increasing its speed when managing the forward command. The whole system was then co-simulated, including Charlie's modifications.

Charlie observed a beneficial speed-up in the exploration. However, he also observed an increasing number of collisions. Charlie, not being an expert on this topic, he created an issue about the collision problem. Both Alice and Bob checked if the problem was on their side. Alice observed in her logs that the obstacles were successfully detected and notified. Bob observed in his logs that the avoidance strategy was defined and executed properly as originally planned, but that sometimes the car collided as if the robot did not see the obstacle on time. Alice confirmed that it was not due to an obstacle detection problem. Alice contacted Charlie and asked him to check that the stop command she sent was correctly realized. Charlie answered that yes, the power on each wheel was set to 0 less than 10ms after the command was sent.

All together, they started investigating the global logs to understand the problem. Since the logs were extensive, it took time, but after a deep investigation, they understood that despite the robot being asked to stop on time, the inertia of the robot was delaying the actual stop, causing it to sometimes collide. As a fix, Alice simply increased the distance threshold used to detect an obstacle (in the Dynamic Detection part). After this fix, the robot speeds up the exploration with no collisions.

This problem appeared because there is a relationship between the speed of the robot and the time required to immobilize the robot, due to inertia. This factor should be taken into consideration

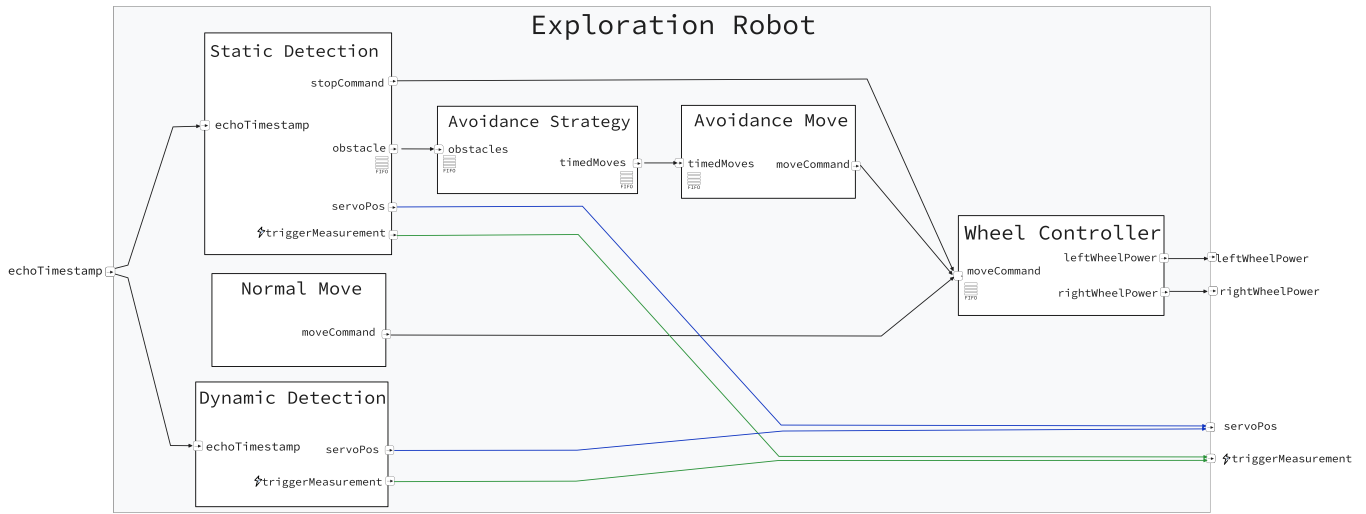


Figure 1: High level model of the system using Block Diagram

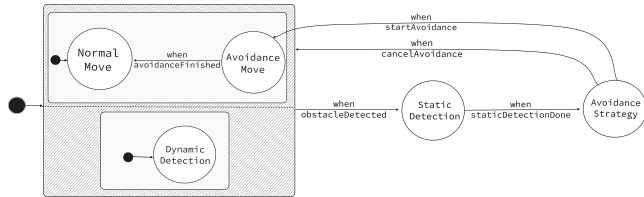


Figure 2: High level coordination of system parts using State Charts

when choosing the distance threshold for the dynamic detection of obstacles. Later in the development process, drawing on their experiences, each time Alice wanted to change this threshold, she notified Charlie so that he could adapt the wheel power; and vice versa. Since Alice is also in charge of developing the static detection, which is triggered when the robot is immobilized, she added a delay between the stop command sending and the start of the environment sweep. Of course, the length of this delay is also influenced by the speed of the robot.

This scenario illustrates that, despite many relationships being made explicit in the various artifacts, some (cyber-physical) relationships are missing and stay only in the heads of the stakeholders. Keeping hidden some of these relationships may lead to communication problems and a longer time to delivery. This motivates the requirement for a tool-supported solution for making explicit these relationships during the collaborative CPS development. Existing models and design artifacts shall be enriched with the extra information describing how design choices on design artifacts influence the behavior of other design artifacts, including from other parts.

In the following sections, we introduce our novel concept of *Influence* to address this requirement and we discuss how it can be exploited as part of a CPS collaborative development process.

3 On the Concept of Influences

Before digging into our definition of influence, we first extract from our motivating example a set of requirements that must be addressed:

Enriching existing models (R_1)

Beyond functional exchanges, it shall be possible to enrich existing models and design artifacts with the extra information of influence making explicit how design choices on design artifacts influence the behavior of other design artifacts.

Motivation: CPS development teams typically use classical software engineering methods, languages and models to specify the behavior of the different parts of a system. Agreements and decisions are made to define the right abstraction and granularity level for their models with respect to their objectives. As a result of this abstraction, these models may overlook some interplay between design choices when various stakeholders are collaborating through (possibly different and heterogeneous) artifacts. Whilst the functional exchanges between the design artifacts are captured using classical software engineering methods, it is common to have interactions between different domains that go beyond mere functional ones. Considering our example, there is no information about the possible influence of changing the power on wheels onto the rest of the system.

Cross design parts and domains (R_2)

It shall be possible to define influences that span across multiple system parts from different domains of expertise and possibly from different stakeholders. *Motivation:* CPS are composed of interconnected parts with specific objectives, which are typically designed by different domain experts. To enhance collaboration, it is important to facilitate the understanding of the influence of one designer's choice on their collaborators' parts. In our example, Alice focuses on the domain of detection and does not have details on the movement control and obstacle strategy

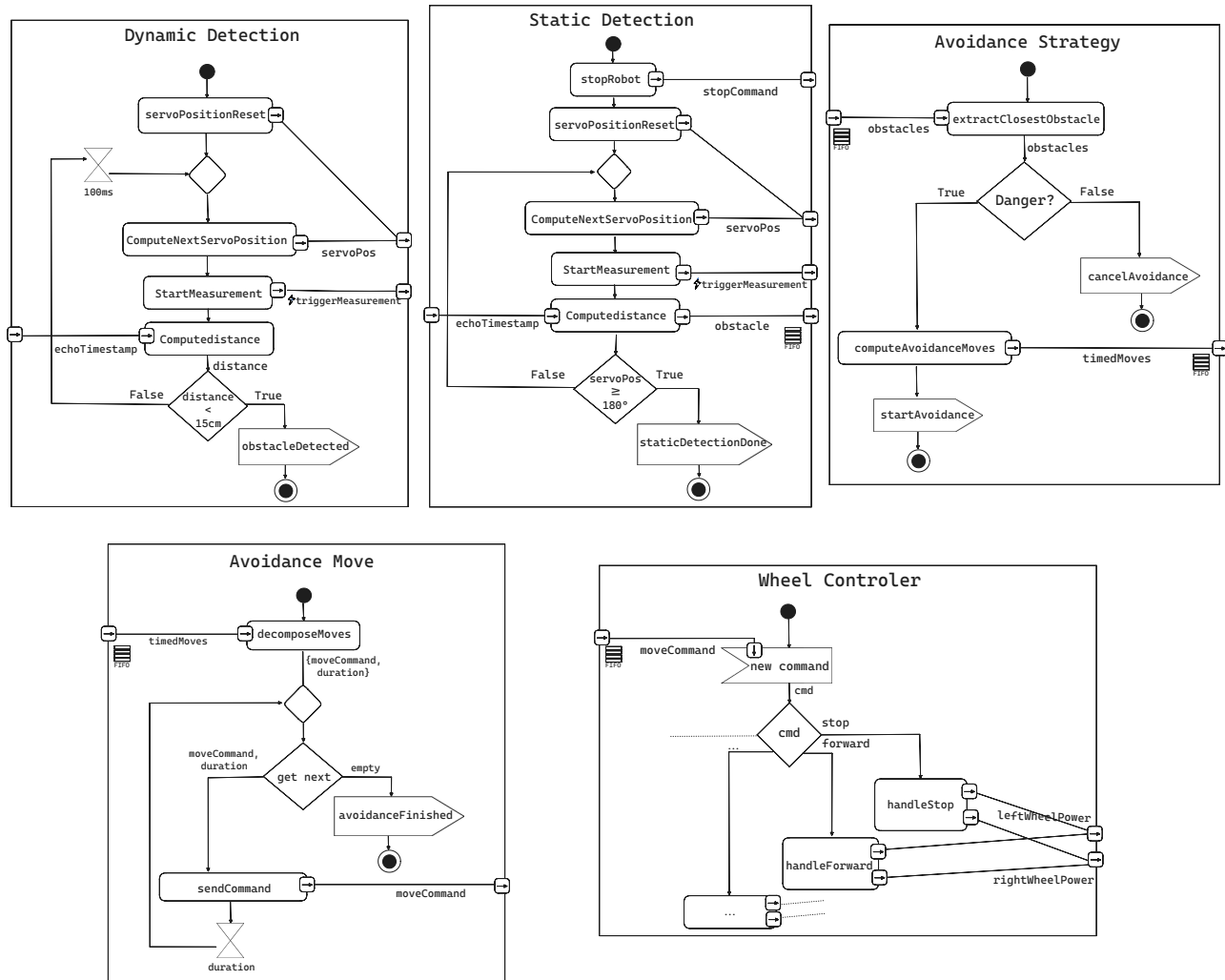


Figure 3: High level model of the system using Activity Diagrams

parts. It would be counterproductive for her to include all the details of other domains in her system specification since it is not part of her expertise and would overcomplicate the system development. As a result, she is unaware of the impact of the changes introduced by Charlie on her part and she has to investigate this manually, e.g., exploring co-simulation logs. Specifying influences between some elements of her part to elements of Charlie’s parts could help Alice in her investigation.

Cyber-Physical interactions (R_3) : Influences shall be able to represent Cyber-Physical interactions between design artifacts.

Motivation: CPS are operating in the physical realm and interactions between the system and its environment can have a critical impact on its proper functioning. An influence may relate not only to software interaction but also

to the interaction between software elements and the physical world. Considering our motivating example, there is a relationship between the power on wheels parameter and the time required to immobilize the robot due to (i) inertia and (ii) the topology of the ground.

Abstraction (R_4) : It shall be possible to define influences at different levels of abstractions and granularity, yet they should contain enough information to enable static and dynamic analysis of the CPS.

Motivation: Depending on the constraints and criticality of an influence with respect to the system requirements, one might want to characterize an influence more or less precisely. In some cases, high-level definitions are sufficient to provide simple feedback to the developer about the impact of his choice. By contrast, in some situations when the influence is critical for the proper functioning

of the system, it is necessary to precisely define it. Over time, the definition of an influence might evolve, e.g., it is refined based on experience and trials. This can be due to several factors : (i) evolving requirements call for a more precise definition, (ii) experience, trials, and co-simulation needs to be performed to iteratively refine the definition of an influence (because experts were not able to precisely define the interaction at first or because it is discovered after execution or simulation of the system). In the context of our motivating example, the impact of inertia can be refined after several co-simulation runs, while the ground topology might remain at a high level of abstraction as it is unpredictable.

Considering the requirements aforementioned, we define an **influence** as: *"the characterization and abstraction of a non fully specified cyber-physical interaction between design choices, which goes beyond the abstractions, models, and specifications identified to address the system requirements."*

It is important to note the distinction with the concept of functional exchange. Functional exchange is a relationship based on functionalities and data exchanges that is intentional and well-documented. In contrast, influences are impacts that one design choice or component can have on another. They are not necessarily (i) identified during design time or (ii) part of the domain. Functional exchanges can lead to the emergence of influences, but influences do not rely on those interactions. Additionally, influences capture behaviors that emerge from the collaboration, which may not be straightforward when designing functional exchanges.

In our exploration of the concept of influence, we have defined characteristics, categories, and metadata of influences. These aim to be identified by or provided to the users. During CPS design, users can partially define an influence between two artifacts, without being certain about its impacts. For example, they may suspect that their system part is related to another one, but be uncertain about what the impact is or how to represent this impact. They can state that uncertainty about the influence, and represent what they know about that. Through collaborative simulation, it is possible to validate or contradict this influence and further explore its semantics. Moreover, it is possible to identify influences after observing the emerging behavior during simulation.

To provide a clear understanding of influences, Figure 4 represents examples of identified influences during the development of the mobile robot use case. Those examples will be referenced throughout the following subsections.

3.1 Characteristics of Influences

3.1.1 Endpoints. An influence has two endpoints: a source and a target; a change in the source will influence the target. Considering requirements R_2 and R_3 , possible endpoints include design artifacts, parts of the system, and elements of the physical environment (i.e., natural factors and other systems not in control of any users of the CPS under development). The notion of endpoint provides a means to (i) clearly pinpoint the elements involved in the influence and (ii) support the notion of directionality of influence (cf. 3.1.2). In Table 1, we exemplify sources and targets within our use

case. For example, we observed influences between the parameters DistanceThreshold (source) and MotorSpeed (target), as well as between the hardware artifact UltrasonicSensor (source) and the part Avoidance Strategy (target). The ambient noise (source) influences the quality of the ultrasonic sensor measurement (target), and the ambient wind (source) influences the movement part (target) of the robot.

In our case study, we did not propose influences targeting the environment in which the robot navigates. Nevertheless, such influence could exist, for instance, a requirement can be related to the sound level in the room, and different configurations of the speed parameters can lead to the violation of such requirements.

Table 1: Source and target definitions

Source/Target	Artifact	System part
Artifact	DistanceThreshold and MotorSpeed	UltrasonicSensor and Avoidance Strategy
System part	Dynamic Detection and UltrasonicSensor	Avoidance Strategy and Wheel Controller
Environment	Noise and Ultrasonic-Sensor	Wind and Wheel Controller

3.1.2 Directionality. We observed that influences typically have a directionality from a source to a target, i.e., one element influences another. We consider bidirectional influences when two elements interact with each other, each influencing the behavior of the other. In such cases, each endpoint is considered as both a source and a target. An example of bidirectional influence is the influence between Motor Speed and Detection threshold, as illustrated in Figure 4. If the speed of the robot is changed, the detection threshold must be modified. The same for the other way around, if the detection threshold is modified, the Motor Speed needs to be adjusted accordingly.

Classifying the directionality is important to determine how to perform integration and collaboration within the CPS development. An unidirectional influence indicates a clear flow of impact, allowing for a straightforward system integration. A bidirectional influence requires a more careful consideration, especially when the influence exists between two different system parts. Characterizing these influences helps to understand how design changes propagate in the system, and, consequently, improve system troubleshooting.

3.1.3 Influence Semantic. It is crucial to explicitly define the semantics of each influence as this will be key to statically or dynamically analyze the behavior of the system and the impact of a design choice. It provides a deeper comprehension of an influence, more than its identification, but detailing how the source impacts the target. Following requirement R_4 , this can be expressed in a continuum from the highest to the lowest level of abstraction, for instance using rules, conditions, constraints, etc.

Influences can, for instance, be expressed using algebraic expressions and defined boundaries of parameters and components. The influence between the Motor Speed and the Detection Threshold

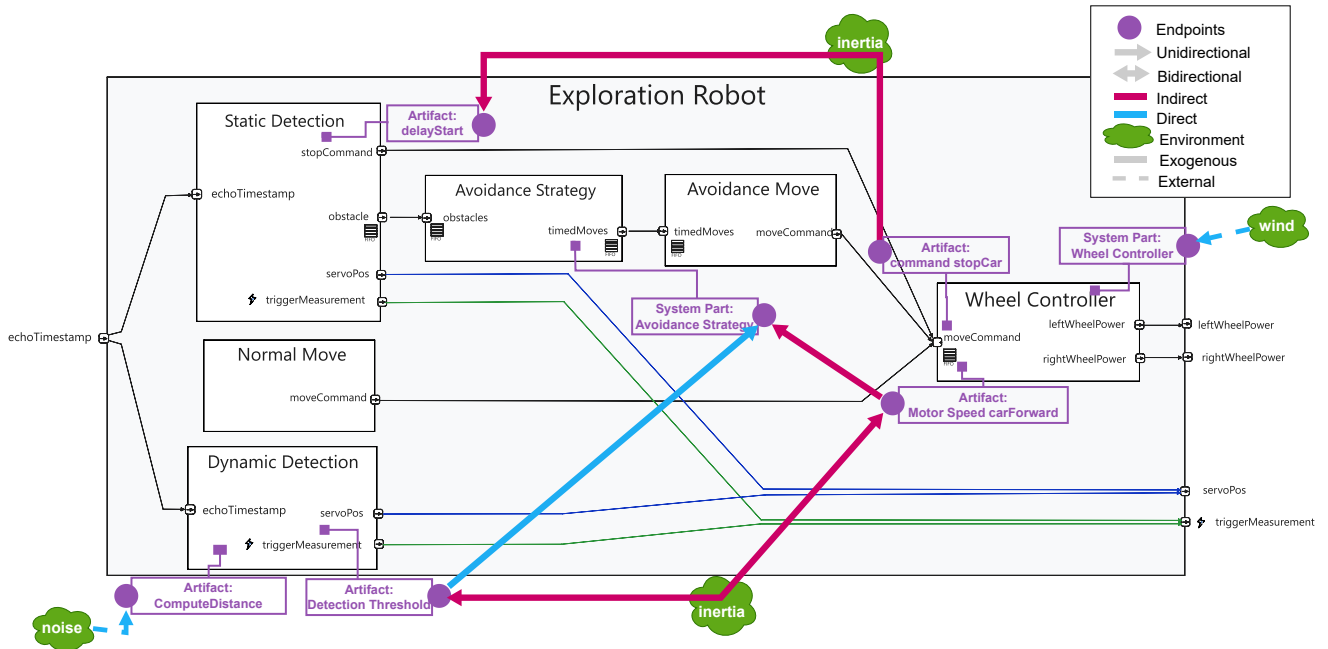


Figure 4: Identified Influences in the autonomous mobile robot use case

can be quantitatively expressed with a function, such as $D = f(S)$, where D is the detection threshold distance, and S expresses the motor speed. This mathematical representation helps in characterizing how changes in the motor speed influence the detection threshold, providing a clear understanding of their impact and establishing allowed ranges for adjustments.

3.2 Categories of Influences

3.2.1 Direct and indirect influences. A direct influence depicts the relation between a source and a target without any intermediary. One element directly impacts another, without involving other elements; therefore changes in one influence the behavior of the other. An influence is considered indirect when the interaction between the source and the target involves a physical phenomenon related to the environment in which the CPS is operating. As shown in 4, an example of such influence is the relation between the Motor Speed and the Detection Threshold for which inertia needs to be considered. It is important to distinguish between these two type of influences. An indirect influence implies that part of the interaction between the source and target (i.e., the environment) (i) is not fully under the control of the development team, (ii) can vary depending on the context of the system (e.g., different locations will lead to different interactions), (iii) is unpredictable, all possible changes cannot be anticipated at design-time.

3.2.2 Endogenous, exogenous and external influences. Endogenous influences emerge between design artifacts of the same system part. The users who work on that part are experts and in control of the parameters, algorithms, and overall design choices. Exogenous influences emerge when they are between design artifacts from different parts. The users do not work on the same system

part, and could even work in different organizations. They are not necessarily experts in the domain of both the source and target parts. External influences happen when the source or the target of an influence is not part of the CPS under development. It can involve environment factors, or other systems not in control of the users. Sometimes, those factors are not included during modeling due to their complexity. So, it may exist an influence between two artifacts, caused by an external element that can vary according to context and is not expressed by a functional exchange within the system.

As illustrated in Figure 4, the influence between the design artifacts Detection Threshold from the Dynamic Detection system part and Motor Speed from the Wheel Controller part is exogenous. The influence between the environmental factor Noise and the Compute Distance, in which the ambient noise influences the proper functioning of the Ultrasonic Sensor is external.

This categorization of influence is particularly relevant to identify the stakeholder whose work might be impacted by an influence and to adapt the feedback accordingly. In case the influence is endogenous, it will be defined by an expert in the domain (both source and target are from the same domain) and the influence can be used to provide detailed and domain-specific feedback on design choices. Exogenous influences are beneficial for promoting a better integration between different system parts. In the case of exogenous influences, and by contrast with endogenous influences, different domain experts are impacted and feedbacks need to be tailored for their common understanding. External influences imply that the physical environment surrounding the CPS is involved and thereby all the uncertainties pertaining to CPS must be considered.

3.3 Metadata of Influences

As detailed in requirement R_4 , an influence can be more or less precisely defined, thereby representing the underlying Cyber-Physical interaction with different degrees of fidelity. To leverage influences in an informed way when analyzing the system and to understand the impact of certain design choices, it is important to provide indications on the fidelity of the influences definitions. To do so, we introduce two metadata that can be attached to an influence in order to characterize the degree of fidelity of an influence. Further metadata could be added in the future.

3.3.1 Confidence. Confidence is an indicator of the degree of fidelity of an influence semantic definition. It can be determined by the users, according to their expertise in the system under development. For example, users with experience contributing to CPS development can provide an initial judgment of their confidence levels for the identified influences. This confidence level can be further validated or refined via co-simulation. Through co-simulation, the emergent system behavior can confirm the initial confidence levels determined by the users, or contradict them, providing a validated understanding of the system's influences.

3.3.2 Likelihood. We define likelihood as the probability that the source of an influence will impact the target. This is particularly relevant in the case of influences with low confidence level. It can be determined by different methods, indicating how often the influence was observed to happen in collected data or while co-simulating. It can be identified that an influence is frequent, and expected to always emerge in the system behavior, even with different design choice configurations. For example, the influences between car speeds and time designed for movements were always observed that in all the recorded occurrences. Otherwise, if the influence is barely observed, or only observed in specific operational contexts, it can be classified as unlikely.

This information are important for decision making, the confidence and likelihood metadata can be used to drive CPS behavior analysis as influences with higher confidence and likelihood are more expected to be root causes of issues than others with lower quality levels.

3.4 Discussion

In this section, we depicted the concept of influences, its categories, characteristics, and metadata. These definitions are essential for the improvement of decision-making during system development. Some influences can be more easily detected, and the users may be aware of them, due to experience and observations during simulation or experimentation. However, in a context where many fields of expertise are involved, it can be challenging to have them in mind when developing. A system may have thousands of design artifacts, spread in different domains of expertise and users. Therefore, explicitly identifying the influences can facilitate managing this complexity.

In addition, since the concept of influences goes beyond functional exchanges, it may be unclear for the users how elements can influence each other. For example, if two artifacts are separated by

different system parts, and many functional exchanges are happening in the system, some influences will be not noticed, and their impacts will not be considered as important as they should be.

With this enriched understanding of influences, we can anticipate potential issues, allowing for preemptive adjustments early in design time. For example, when Charlie tries to commit the change of the speed variable, he receives a notification, stating that his change is not recommended because it may impact other parts of the system. Moreover, the enriched semantics can help deepen the system understanding after co-simulation. The resulting outcomes can be analyzed based on the identified and characterized influences. By integrating into automated analysis tools, it could provide not only insights into the identification of root causes, but also an understanding of how this root element caused the issue. For instance, it could provide Bob with the influence path from the source of the problem, detailing the characteristics of each influence, and indicating possible solutions to mitigate the problem.

4 Conceptual Workflow for Collaborative CPS Development

The definition of influences can be explored in different stages of the system development process. In Figure 5, we represent the envisaged workflow. We omitted other phases that are typically part of the development process, such as Requirements Engineering, Planning, etc., to focus on the core aspects of our contribution.

The proposed workflow addresses two main purposes. Primarily, we introduce a static analysis to provide early feedback on the system behavior during the design phase. It is termed static because it has the purpose of examining influences in the stage of design without running an execution (simulation or proper execution) of the CPS but solely by analyzing statically the models. By analyzing influences in this early stage, it is possible to detect potential issues before their propagation in the development process. Secondly, we incorporate a dynamic analysis to provide feedback based on the resulting system behavior observed during simulations, experimentation, or real-life operations. With the second analysis, we aim to enhance the understanding of the influences and discover new ones within the system.

4.1 Leveraging influence in the Collaborative CPS design

As shown in Figure 5, in the stage of **Artifact design**, the developers define the initial system models, according to their individual development process. Following that, Semantic Enrichment is realized. During this phase, the same users or experts (e.g. system integration engineer) can identify and define influences by exploiting their expertise and experience. They can characterize and add metadata to the influence according to their understanding of the system behavior. For instance, they might be confident in the existence of an influence in the system, based on the development of the same or similar CPS, or they may suspect its existence. Similarly, they may be aware of the existence of an influence, but do not know its precise definition. In such case they will define new influences whose semantics are abstract with a low level of confidence. This

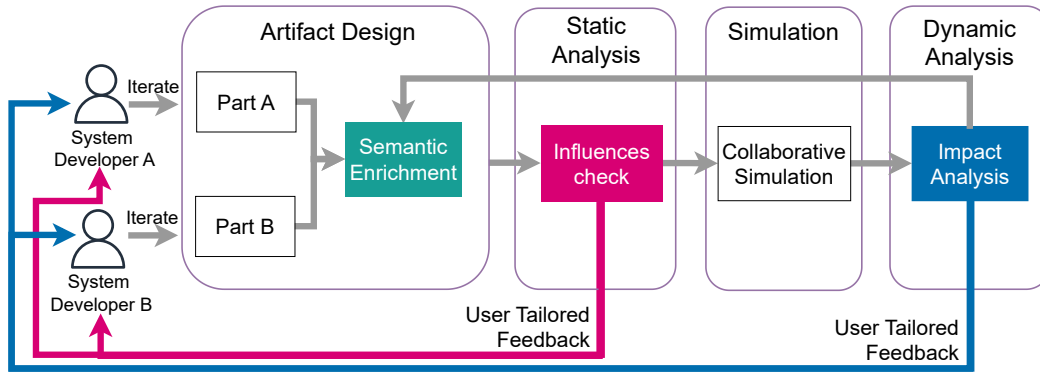


Figure 5: Integration of our envisaged workflow on System Development Pipeline

process of documenting the current knowledge ensures that a maximum of potential influences are considered, even if the details are still not clear.

Additionally, it is essential to integrate the models with a versioning control system, to track and maintain a documented history of design changes and influences. This ensures that the evolution of influences corresponding to the design changes is recorded. The users can revisit and understand why certain changes were made during development, and evaluate the occurrence of origin of new influences due to modifications.

The system will pass through a **Static Analysis** stage. This stage aims at evaluating the impact of influences and checking whether this leads to the violation of some of the system’s requirements and providing feedback to the users accordingly. Metadata, characteristics, and categories attached to the influences are exploited to refine the analysis results. It may happen that the configuration of an artifact impacts related artifacts, requiring adjustments of those artifacts to ensure requirements are fulfilled, being in the level of parametrization. Additionally, a design choice may not only impact related elements but also provoke the origin of other influences within the system that were not initially considered in the previous phase, with impacts being more on a structural level. These issues may require more significant adjustments and further investigation, which could lead to the need of Dynamic Analysis.

This stage includes techniques that compare the initial and new versions of the models, identify the modifications, and check if the design changes made could provoke any conflict. For example, if a developer attempts to modify a design artifact, the impact of this change on the system will be analyzed, based on the influences defined previously in the Semantic Enrichment stage. The concerned developers are notified and informed about the outputs of the analysis. This may involve developers from different domains, so the feedback must be personalized, providing them with details according to their field and role in the CPS development. According to the feedback, they can evaluate the potential issues and collaborate within the involved developers of the CPS to discuss and find solutions for identified conflicts.

4.2 Leveraging collaborative simulation for deepening system behavior understanding

The second part of the proposed workflow involves the exploration of collaborative simulation outcomes. It is crucial to observe the system behavior during runtime, as it provides insights into the actual behavior of the system and its evaluation.

During the **Simulation** stage, the multiple system parts are co-simulated, ensuring that data are exchanged and the events are managed between individual simulators. Once the collaborative simulation is performed, the **Dynamic Analysis** stage starts. The simulation outputs and the known influences will be analyzed for the comprehension of the system behavior. The objective is twofold : (i) to refine the knowledge about existing influences and possibly discover new ones, and (ii) to identify new possible violations of requirements which could not be identified by static analysis only. This involves techniques that can reason on simulation logs. A possible method is to change parameter values and vary other design artifacts, perform co-simulation for each of them, and empirically derive influences and their semantics. In case a user identified influences between two design artifacts with high-level semantic definition, this method can suggest refining it, for instance providing mathematical representations and constraints. Another interesting investigation would be to improve the identification of the endpoints of an influence. For example, if the users identified influences between two system parts, but do not know which specific design artifact is responsible for the impact and how it happens, the Dynamic Analysis could provide more information on those details. Multiple co-simulation runs can also provide a quantitative evaluation of the system. This evaluation can provide metrics for understanding the observed impact of the influences on system behavior. Metadata such as the influence’s confidence and likelihood levels can be derived.

Another potential investigation is the analysis of issues in co-simulation results. Through Dynamic Analysis, developers can be provided with feedback on the possible root causes. The unsatisfactory outcomes can be evaluated in the context of the identified influences, and developers can be provided with insights into which specific artifacts are responsible, for facilitating troubleshooting. Moreover, it can improve collaboration, providing personalized

feedback to the identified developers concerned in resolving the issue, according to their roles in CPS development.

With Dynamic Analysis, users will be able to make informed iterations in the system model and improve the definition and characterization of the influences, overall enhancing the system design and integration. In the following iterations, the influence checks will be more accurate, the users can identify errors even before the simulation phase. This can reduce system development time, and optimize the use of resources for simulating and prototyping.

5 Related work

In recent years, several surveys have been conducted to investigate the semantic relationships existing between artifacts in CPS development.

Traceability plays a central role in this investigation, in which it is possible to define traces (source and target), enabling the tracking and management of artifacts. In a collaborative system, traceability is a remarkable challenge, since many stakeholders are involved and are part of different organizations [14]. Capra [8] and Cameo Systems Modeler [2] are some examples of tools that offer traceability management that links requirements to system components, test cases, and other models.

However, it is usually a traceability of the static structure of the system, and the dynamics of the system are not explored and explicitly expressed. In [9], related to software systems, links between behavioral safety requirements to SysML diagrams, presented an algorithm for providing users narrowed view of links related to a given requirement. In [1], focused on Cyber-Physical Systems, provides links between system goals, requirements, and SysML diagrams, and provides consistency tests to exploit the direct and indirect relationships between artifacts. This work aims to take a step further by exploring deeper details on influences within the system, providing deeper details on both static and dynamic analysis of influences.

Another area in the state of the art is related to tools providing **Impact Analysis feedback** for the users after collaborative simulation. The INtegrated TOolchain for Cyber-Physical Systems [7] is a notable framework. In addition to realizing traceability, it utilizes Design Space Exploration (DSE) for finding alternative designs by ranging parameters and performing co-simulation of each configuration. [11] proposed an architecture for supporting tactical and operational decisions exploring Design of Experiments (DOE). This approach provides optimal solutions for each scenario, combining them into a design matrix, so the users can make informed decisions based on expected outcomes. While these works aim to find the optimal parameters with co-simulation, our objective contribution is to provide feedback to improve the understanding of the system behavior, improving the semantics of the existing influences and discovering new ones.

PROSTEP IVIPSoftware [10] is a relevant work that includes traceability and simulation analysis capabilities. Their approach includes documentation about artifacts through the development cycle, which is used for traceability. Following that, they evaluate the risks from the artifact choices based on simulation results, classifying the critical level of those decisions. In our work, we intend to use simulation results differently, evaluating influences

and identifying new ones, and providing feedback about likelihood, and confidence level.

In light of the discussed state of the art, this work aims to address the limitations of system behavior understanding. Traceability and impact analysis were extensively explored in recent years, however, gaps remain in understanding the influences within the system. By leveraging simulation results, our proposed approach seeks to provide a deeper knowledge of system influences, which leads to more informed decision-making in complex, collaborative system development environments.

6 Conclusion

This paper has introduced the concept of influences as a means to enhance collaborative development. Specifically, we defined the concept of influences as the characterization of non fully specified cyber-physical interaction between design choices, going beyond the abstractions identified to address the system requirements. We proposed classifications, categories, and metadata of influences. By exploring this concept, we aim at enriching system design models for improving decision-making processes. We also propose a conceptual workflow, that leverages the concept of influence to include Static Analysis, investigation at design time, and Dynamic Analysis, investigation after execution of co-simulation, as part of a CPS development process. Our work is in the conceptual stage, and future work will focus on implementing the proposed workflow, considering more complex use cases, and further refining our approach to ensure efficient Collaborative Cyber-Physical System development.

7 ACKNOWLEDGEMENTS

The research leading to this publication has partially received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreements 101070455 (DYNABIC).

References

- [1] Amal Ahmed Anda, Daniel Amyot, and John Mylopoulos. 2023. Traceability Management of Socio-Cyber-Physical Systems Involving Goal and SysML Models. *Modelling* 4, 2 (2023), 133–167.
- [2] Dassault Systèmes. 2023. Cameo Systems Modeler. <https://www.3ds.com/products/catia/no-magic/cameo-systems-modeler>. Accessed: 21/06/2024.
- [3] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni Vincentelli. 2012. Modeling Cyber-Physical Systems. *Proc. IEEE* 100, 1 (2012), 13–28. <https://doi.org/10.1109/JPROC.2011.2160929>
- [4] DFRobot. n.d.. Cherokey 4WD Mobile Platform. https://wiki.dfrobot.com/Cherokey_4WD_Mobile_Platform_SKU_ROB0102_. Accessed: 01/07/2024.
- [5] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. Co-Simulation: A Survey. *Comput. Surveys* 51, 3 (May 2018), 49:1–49:33. <https://doi.org/10.1145/3179993>
- [6] Edward Griffor, Christopher Greer, David Wollman, and Martin Burns. 2017. Framework for Cyber-Physical Systems: Volume 1, Overview. <https://doi.org/10.6028/NIST.SP.1500-201>
- [7] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzon, Jörg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, and Andrey Sadovykh. 2016. Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. 1–6. <https://doi.org/10.1109/CPSData.2016.7496424>
- [8] Salome Maro and Jan-Philipp Steghöfer. 2016. Capra: A Configurable and Extendable Traceability Management Tool. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 407–408. <https://doi.org/10.1109/RE.2016.19>
- [9] Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, and Thierry Coq. 2012. A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Information and Software Technology* 54, 6 (2012), 569–590.

- [10] prostep ivip Association. 2021. White Paper - SmartSE - Simulation-based Decision Making and Release. https://www.prostep.org/en/medialibrary/translate-to-english-detail?ai%5Baction%5D=detail&ai%5Bcontroller%5D=Catalog&ai%5Bd_name%5D=wp_smartse_sdmr&ai%5Bd_pos%5D=. Accessed: 22/06/2024.
- [11] Shady Salama and Amr B. Eltawil. 2018. A Decision Support System Architecture Based on Simulation Optimization for Cyber-Physical Systems. *Procedia Manufacturing* 26 (Jan. 2018), 1147–1158. <https://doi.org/10.1016/j.promfg.2018.07.151>
- [12] Martin Törngren and Ulf Sellgren. 2018. Complexity Challenges in Development of Cyber-Physical Systems. In *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, Marten Lohstroh, Patricia Derler, and Marjan Sirjani (Eds.). Springer International Publishing, Cham, 478–503. https://doi.org/10.1007/978-3-319-95246-8_27
- [13] Itorobong S. Udoh and Gerald Kotonya. 2018. Developing IoT applications: challenges and frameworks. *IET Cyber-Physical Systems: Theory & Applications* 3, 2 (2018), 65–72. <https://doi.org/10.1049/iet-cps.2017.0068> _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cps.2017.0068>.
- [14] Rebekka Wohlrab, Eric Knauss, Jan-Philipp Steghöfer, Salome Maro, Anthony Anjorin, and Patrizio Pelliccione. 2020. Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture. *Requirements Engineering* 25, 1 (2020), 21–45.