Utilizing Sensor and Actuator Virtualization to Achieve a Systemic View of Mobile Heterogeneous Cyber-Physical Systems

Martin Richter[1], Reinhardt Karnapke[2] and Matthias Werner[1]
[1]*Operating Systems Group, Chemnitz University of Technology, Germany*
[2]*Chair of Distributed Systems/Operating Systems, Brandenburg University of Technology, Germany*

Keywords: Cyber-Physical Systems, Virtualization, Mobility, Heterogeneity, Transparency.

Abstract: When programming cyber-physical systems, application developers currently utilize physical sensors and actuators individually to achieve the desired observations and impacts within the physical world. This is an error-prone and complex task given the size, heterogeneity, and mobility of prevailing cyber-physical systems. We introduce an application model that allows the application developers to take a physical perspective. By means of this model, the programmers describe desired observations and influences with respect to the physical world without directly referencing physical devices. We present an additional model for a runtime environment that transparently utilizes the available physical devices to reach the application developers' targets. We show that an implementation of our models is functional via simulation.

1 INTRODUCTION

Cyber-Physical Systems (CPS) play an ever-increasing role because of trends like the Internet of Things and Industry 4.0. Such systems observe and influence their physical environment via sensors and actuators. These devices are distributed through space and may be heterogeneous as well as mobile. Classically, the application developers program these devices individually to achieve a desired impact on the physical world. This alone, is an error-prone and complex task under the mentioned circumstances due to explicit communication between and coordination of the devices. Adding to this, it is the standard for programmers to take a digital perspective when programming such systems. This makes the desired impact on the environment less clear, as control and measurement signals may not reflect physical circumstances accurately. For example, the input signal for a light actuator can be represented either by `ON` or `OFF` while the resulting illumination is not clear from setting this signal.

We argue that with the decreasing costs of computation units and their increasing processing power, more sophisticated models should be employed. We introduce an application model that allows the application developers to take a physical perspective when programming CPS. It allows them to describe the properties of physical phenomena of interest and how these properties should evolve over time. This makes the desired impact of the application on the environment clear. Complementing this application model, we introduce a Runtime Environment (RTE) that supports the interpretation of the application programmers' descriptions. Based on these descriptions the RTE utilizes the available physical sensors and actuators transparently with respect to the application. We achieve this by integrating device virtualization into the RTE. We define virtualization as the deployment of a virtual device that, during runtime, is mapped onto (possibly multiple) physical devices on demand. Our model refers to virtual sensors and actuators that represent the capabilities to observe and influence the applications' phenomena of interest. During runtime, each virtual device utilizes varying sets of physical devices based on their capabilities to achieve the desired observation or influence of the physical phenomena of interest. As a consequence, the application is detached from directly interacting with the physical devices. This in conjunction with programming from the physical perspective allows the developers to take a systemic view that is independent from changing sets of mobile and heterogeneous physical devices. Thus, distribution transparency is achieved and the programmers are able to focus on their targets with respect to the physical environment.

The remainder of this paper is structured as follows. Section 2 presents related work focussing on

Richter, M., Karnapke, R. and Werner, M.
Utilizing Sensor and Actuator Virtualization to Achieve a Systemic View of Mobile Heterogeneous Cyber-Physical Systems.
DOI: 10.5220/0012715800003758

In *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024)*, pages 207-214
ISBN: 978-989-758-708-5; ISSN: 2184-2841

sensor and actuator virtualization for CPS. In Section 3, we provide a running example for illustrative purposes. Section 4 contains the theoretical concept of our approach and presents an initial evaluation. Section 5 provides a conclusion and an outlook on the next steps to take.

2 RELATED WORK

Existing operating systems for robotic systems, such as the *Robot Operating System 2* (*ROS 2*) (Macenski et al., 2022) or *XBot2* (Laurenzi et al., 2023), are fundamentally suited for deployment in CPS. Though these operating systems provide basic abstractions for utilizing various individual devices, they lack the transparent utilization of sets of devices that our virtualization model offers.

In (Tsiatsis et al., 2010), an architecture is introduced that utilizes a resource layer. This layer abstracts from individual physical devices and provides a uniform interface for accessing them. Apart from this interface, the developers have to specify which sensors and actuators are to be employed. This requires explicit knowledge of the capabilities of the devices. As a result, the application is still directly bound to physical sensors and actuators.

In (Suh et al., 2013), a similar concept is presented. It enables the developers to select physical sensors and actuators based on their capabilities and the contexts they are located within. After the programmers select the devices, they have to control them explicitly. Therefore, no virtualization is achieved which inhibits transparently utilizing multiple different sensors and actuators.

In (Fernandez-Cortizas et al., 2023), an approach to software synthesis is presented that transforms an application into a behavioral plan consisting of multiple actions. Before runtime, this plan is distributed among the various physical devices based on their capabilities. Each device possesses its own implementation of the different actions. As the distribution of tasks is performed before runtime, no dynamic task allocation is possible and changing environmental factors cannot be taken into account.

In (Vicaire et al., 2010a) and (Vicaire et al., 2010b), the concept of *Bundles* is introduced. It enables the developers to define groups of physical devices that are utilized for the execution of the application. Each group is identified by an abstract device type (e.g., cameras). The group membership of a device may change during runtime based on changing conditions (e.g., motion). The devices within the group are managed individually by the developers. Consequently, there is no virtualization support for heterogeneous devices.

In (Ni et al., 2005) and (Borcea et al., 2004), a similar approach is chosen. The application selects devices based on their location and capabilities. After selection, the developers manage sensors and actuators individually which is in conflict with a transparent utilization of multiple physical devices.

In (Seiger et al., 2015), *ROS 2* is extended by an abstraction layer that focuses on the uniform programming of home robots. Each of these robots may possess different abstract capabilities (e.g., grab or move). The robots may implement these capabilities differently based on their available sensors and actuators. The application utilizes abstract capabilities to describe the desired behavior of a robot. As the robots are programmed individually the concept neither provides distribution transparency nor virtualization of multiple devices.

In (Beal and Bachrach, 2006), an approach is presented that allows programmers to develop applications with respect to continuous regions in space. A virtual device is present at each location within these regions. Every virtual device provides measurements, processing power, and actuation. The execution of the application is then approximated by a discrete set of physical devices that are located within the region. Each of these devices executes the same application code. This code consists of operations that refer to the local state of the executing node as well as the states of nodes in its proximity. This allows the developers to take a systemic view but limits the system to a homogeneous set of devices.

It is evident from the discussed concepts that existing approaches focus on making physical devices more accessible for application programmers. Though this takes away complexity from developing applications for CPS, key challenges like the utilization of multiple heterogeneous physical devices remain open.

3 RUNNING EXAMPLE

This section presents a running example that shows the strengths of our approach with respect to the detachment of the application programmers from the utilized physical devices.

The example comprises a factory scenario in which light-sensitive products are fabricated in a series of steps. Each step takes place in a different factory hall. Hallways connect the buildings and autonomous robots move the products between them. These robots avoid regions exposed to light by uti-

lizing brightness sensors. To keep their path planning as efficient as possible, light sources should be turned off within the factory during production. This cannot generally be guaranteed as human workers irregularly and sparsely have to perform maintenance on the fabrication machines. The application in this scenario comprises the lighting control within the factory such that working conditions are safe for humans while influences on the products are minimized. This encompasses individually turning lights on or off depending on where people are present. To identify people within the factory, the application may utilize information from security cameras as well as from the employed robots (e.g., attached infrared cameras and distance sensors).

Our virtualization model allows the application developers to focus on describing their target, i.e., a desired observation of and influence on the physical environment. Transparent to the application, the RTE handles the technical details for utilizing the required devices (e.g., the available cameras, lighting, robots, and their positioning). The running example encompasses mobility and heterogeneity of devices, physical phenomena, as well as different physical contexts that restrict the capabilities of physical sensors and actuators. Thus, it is well suited for presenting the capabilities of our virtualization model.

4 CONCEPT

As described in Section 1, the developers take a physical perspective such that a systemic view is accomplished. The following sections introduce an environmental model and an application model to describe the programmers' view on the physical environment. Based on these models, we introduce an approach to I/O-device virtualization. It connects the application and the physical devices that influence and observe physical phenomena.

4.1 Environmental Model

The developers create an application for observing and influencing a physical phenomenon of interest P within the physical system Σ. The system region is denoted X_Σ and represents the developers' space of interest within which the physical phenomenon may reside (e.g., a factory comprised of different production halls and hallways). Multiple properties characterize such a phenomenon (e.g., the shape of a human and the brightness around it). These properties represent the phenomenon's state $\vec{z}$ and may change over time t. The change of state $\dot{\vec{z}}$ represents the phenomenon's behavior. It is composed of an internally induced change $\dot{\vec{z}}_{int}$ and an externally induced change $\dot{\vec{z}}_{ext}$. The function $\tilde{f}$ represents their composition: $\dot{\vec{z}}(t) = \tilde{f}(\dot{\vec{z}}_{int}(t), \dot{\vec{z}}_{ext}(t))$. Internal factors resemble a change of state $\dot{\vec{z}}_{int}$ based on the phenomenon's current state $\vec{z}$ (e.g., an object changing its position due to its current velocity). External factors refer to changes $\dot{\vec{z}}_{ext}$ induced by controlled activities of actuators (e.g., an object changing its position due to a robot applying force to it). Control signals $\vec{u}$ induce these actuator activities. The function f describes the behavior of the phenomenon based on its current state $\vec{z}$ and the actuator control signals $\vec{u}$ (similar to control theory): $\dot{\vec{z}}(t) = f(\vec{z}(t), \vec{u}(t))$.

Sensors S provide information on the properties of a phenomenon by measuring physical quantities q (e.g., a camera measures electromagnetic radiation). Actuators A influence these properties by inducing changes $\dot{q}$ in physical quantities, (e.g., a light source illuminates its environment). Sensors and actuators in conjunction form the available physical I/O devices D within the system. This set may change over time due to devices joining or leaving the system (e.g., due to failure and repair). In (Richter et al., 2023) we introduced a physical context model for sensors. The same model is applicable to actuators as well. It allows to infer how the outputs of devices may be utilized with respect to their capabilities and the physical contexts they reside in. The model associates with each device d a class ζ, a context region $X_c(d)$, an effective output region $X_{eff}(d)$, an associated physical quantity $d.q$, and a location $d.\vec{x}$.

4.2 Application Model

As described in Section 1, the developers write the application from a physical point of view. It is their goal to determine sufficient external influences on physical phenomena of interest, such that these phenomena reach a target state. Therefore, their application is composed of a phenomenon description. Such a description consists of a state description vector $\vec{\rho}$ that allows to observe the physical phenomenon's state $\vec{z}$ based on interpretations of available sensor measurements, and a behavioral function b that determines the required external state influences $\dot{\vec{z}}_{ext}$ that alter the state of the phenomenon over time via actuator activities while taking the phenomenon's internal state changes into account.

The state description vector $\vec{\rho}$ provides the information necessary for discriminating different phenomena. Each of its elements is a tuple, consisting of a type τ_i and a rule r_i. A type $\rho_i.\tau$ represents a set of possible values for the property. A rule $\rho_i.r$ is a func-

tion that equals one if a value of type τ_i characterizes the phenomenon (e.g., the shape of an object should resemble a human). Within the physical system, possibly multiple phenomena $\underline{P} = \{P_1 \dots P_m\}$ may be present that suffice the description $\vec{\rho}$. The following equation describes this: $\forall P_i \in \{P_1, \dots P_m\} : \vec{z}_{P_i}(t) = \left[z_{P_i,1}(t) \quad \dots \quad z_{P_i,n}(t)\right]^{\mathbf{T}}, z_{P_i,j}(t) \in \rho_j.\tau, \ \rho_j.r(z_{P_i,j}(t)) = 1, \ j \in [1,n]$. The RTE transparently utilizes a function g to instantiate the state vectors $\vec{z}_{P_i}$ for each observed phenomenon that suffices the state description vector. This function interprets the outputs of the available sensors $S(t)$ based on the state description $\vec{\rho}$ while taking the sensors' related output and context regions into account. This allows determining values for the different state variables as well as calculating the spatial regions X_{P_i} in which the phenomena reside, i.e., $\left[(\vec{z}_{P_1}(t),\, X_{P_1}(t)) \quad \dots \quad (\vec{z}_{P_m}(t),\, X_{P_m}(t))\right]^{\mathbf{T}} = g(S(t), \vec{\rho}),\ P_j \in \underline{P}$.

For describing how a target phenomenon state should be reached, the application programmers develop the behavioral function b. Its inputs are provided by the RTE and are composed of the measured state vector $\vec{z}_{P_i}$ that allows the application to consider internal state changes, and a set of currently available external state influences $\dot{\underline{z}}_{P_i,ext}$ that enable the application to choose a vector of desired influences $\dot{\vec{z}}_{P_i,ext}$ from them to reach a target state such that $\dot{\vec{z}}_{P_i,ext}(t) = b(\vec{z}_{P_i}(t), \dot{\underline{z}}_{P_i,ext}(t)), \dot{\vec{z}}_{P_i,ext}(t) \in \dot{\underline{z}}_{P_i,ext}(t)$. The phenomenon may reach a target state without any outside influences (i.e., due to internally induced state changes). Therefore, the application developers have to take the current state $\vec{z}_{P_i,ext}$ of the phenomenon and its internally induced state changes (that are deducible from its current state) into account. Under these considerations, by applying b they select an externally induced state change $\dot{\vec{z}}_{P_i,ext}$ from the available externally inducible state changes $\dot{\underline{z}}_{P_i,ext}$.

The RTE calls the behavioral function b of the corresponding phenomenon description for each identified state vector $\vec{z}_{P_i}$. The RTE transparently determines the currently available externally inducible state changes via a function $\underline{h}$ based on the currently available actuators A and the region X_{P_i} in which the phenomenon instance resides, i.e., $\dot{\underline{z}}_{P_i,ext}(t) = \underline{h}(A(t), X_{P_i}(t))$. Taking the phenomenon's region into account is necessary as the effective output regions of actuators (see Section 4.1) have to intersect the phenomenon's region to influence it. Based on the result $\dot{\vec{z}}_{P_i,ext}$, the RTE transparently utilizes a function $\bar{h}$ to determine sufficient actuator inputs $\vec{u}_{P_i}$ for the available actuators A to influence the state of phenomenon P_i. This is summarized by the equation

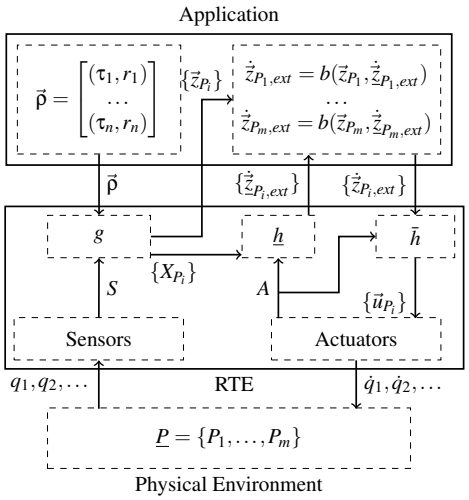


Figure 1: Depiction of the application model, consisting of a state description $\vec{\rho}$ and a behavioral function b for a phenomenon P of which multiple instances P_i may be present; including the interface to the RTE (i.e., g, $\underline{h}$, and $\bar{h}$).

$\vec{u}_{P_i}(t) = \bar{h}(A(t), \dot{\vec{z}}_{P_i,ext}(t))$. Figure 1 depicts our application model in conjunction with the described interfaces to the RTE.

The presented model enables the application developers to describe desired influences on physical phenomena. Depending on the application, there may be a need for a default behavior, i.e., developers specifying desired influences on the environment for locations at which the phenomenon is not observed. In our running example, this refers to lights being turned off at all locations with no worker being present. This default behavior is not dependent on any state description. Rather, it specifies a behavior for all locations that is then overwritten by the behavior of observed non-default physical phenomena at their respective locations. This overwriting is performed by the RTE based on priorities that the application developers assign to the behavior of phenomena. From the application's perspective, the developers achieve this default behavior by specifying a physical phenomenon with no state description (i.e., by only specifying the behavioral function b). Such a default phenomenon represents a phenomenon for which all available state changes may be relevant.

Listing 1 depicts this for our running example in Python code. In the main function, the developers initialize the RTE by making the phenomena of interest (i.e., `Default` and `Person`) known. The `Default` phenomenon specifies that from all available inputs in all locations, the ones that provide a minimum brightness should be chosen (i.e., lights being turned off). The `Person` phenomenon encompasses a state description and a behavior description. The state de-

scription specifies that the phenomenon possesses the shape of a human and that the brightness around it is of relevance (although no direct rule is given as a person should be identified independent of the brightness around it). The behavioral function specifies that from all available property changes that may influence the person, the one that provides the minimum brightness above the required working conditions of 250 Lumen should be chosen. The priorities `P_MIN` and `P_MAX` are assigned such that the default phenomenon is overwritten wherever a person is identified. It is now the task of the RTE to identify these phenomena of interest, determine available influences on them, call the behavioral functions of the instantiated phenomena, and realize the resulting desired influences.

```
import rte
from priorities import P_MIN, P_MAX
from properties import Brightness, Shape
from properties import DefaultPhen, PhysicalPhen

class Default(DefaultPhen):
  def priority():return P_MIN

  def behavior(_, changes):
    if "Brightness" not in changes:return{}
    return{"Brightness":min(changes["Brightness"])}

class Person(PhysicalPhen):
  def priority():return P_MAX

  def state():return{"Brightness":(lambda b:True),
                     "Shape":(lambda s:s==Shape.human)}

  def behavior(state, changes):
    if state["Brightness"]>=250:#Lumen
      return{}
    if "Brightness" not in changes:return{}
    chosen = [c for c in changes["Brightness"] if c>=250]
    if chosen==[]:return{}
    return{"Brightness":min(chosen)}

def main():
  rte.initialize([Person, Default])
```

Listing 1: Python code for the application implementing the desired behavior of our running example (see Section 3).

4.3 Runtime Environment

As described in Section 1, we define virtualization as the transparent provision of a virtual device that is mapped onto physical devices at access or on demand. Based on the application's target, a virtual device dynamically merges the capabilities of possibly multiple physical devices. In our application model, the functions g, $\underline{h}$, and $\bar{h}$ represent this abstractly. These functions respectively aggregate and disseminate information according to the application's goal and the available physical devices. To realize these functions, we utilize a two-layered approach: a virtual device layer for observing and influencing individual phenomenon properties via virtual sensors and actuators, and an observer and controller layer for managing the different virtual devices for the aggregation of the phenomenon state (observer), the determination of available phenomenon state influences (controller), and the dissemination of the desired externally induced state change (controller). The following sections introduce the basic components of the RTE that are developed by the system programmers. This encompasses virtual sensors and actuators, the observer, as well as the controller.

4.3.1 Virtual Sensors

A virtual sensor represents the capability to observe a property of a physical phenomenon. The outputs of single physical devices may not be directly related to such a property. For example, the measurement of a camera (an array of pixels) does not directly provide information on the shape of a physical object. Additionally, depending on the available physical devices, a property may have to be observed by multiple sensors. For instance, at least two cameras are required for performing visual triangulation to determine the position of an object. Thus, a virtual sensor represents the mapping of possibly multiple physical sensor measurements onto a physical property of type $\rho_i.\tau$. To achieve this, the virtual device may utilize a set of different output processes $\widehat{\Pi}$. For example, a virtual sensor for the detection of object shapes may apply camera-based or Lidar-based methods.

Each virtual output process $\widehat{\pi}$ requires different sets of physical devices with varying capabilities. Therefore, the process utilizes a selection function ψ to determine multiple sets $\underline{S}$ of such suitable devices from the set of currently available sensors S. This selection is based on requirements on each device d. These requirements circumvent the assignable physical devices' classes ζ, context regions $X_c(d)$, positioning $d.\vec{x}$, effective output regions $X_{eff}(d)$, and associated physical quantities $d.q$. For example, to perform triangulation the chosen physical devices have to be correctly positioned cameras with overlapping effective output regions. The selection function returns a set of physical device sets, each of which is suitable for the execution of the process, i.e., $\underline{S}_{\widehat{\pi}}(t) = \widehat{\pi}.\psi(S(t))$, $\underline{S}_{\widehat{\pi}}(t) \subseteq \mathcal{P}(S(t))$, $\widehat{\pi} \in \widehat{s}.\widehat{\Pi}$.

Each virtual output process $\widehat{\pi}$ utilizes an observation function ϕ to determine a value $z_{\widehat{\pi}}$ for a physical property of type $\rho_i.\tau$ (see Section 4.2). The outputs of possibly multiple suitable sensors $\underline{s}$ form the input of the observation function. Depending on those devices (i.e., their individual context regions X_c and effectively observed regions X_{eff}), the result $v_{\widehat{\pi}}$ of the output process $\widehat{\pi}$ encompasses the measured property $z_{\widehat{\pi}}$ and a region of space $X_{\widehat{\pi}}$ for which it is valid. For example, when performing interpolation for a set of brightness sensors, the sensors' enclosed region forms

the result region. The following equation summarizes this: $v_{\widehat{\pi}}(t) = (z_{\widehat{\pi}}(t),\, X_{\widehat{\pi}}(t)) = \widehat{\pi}.\phi(\underline{s}),\ \underline{s} \in \underline{S_{\widehat{\pi}}}(t)$. Based on the different sets of suitable devices $\underline{S_{\widehat{\pi}}}$, the observation function produces different results. The set $V_{\widehat{\pi}}$ represents all results for the observation function obtained by utilizing the different device sets in $\underline{S_{\widehat{\pi}}}$, i.e., $V_{\widehat{\pi}}(t) = \{\widehat{\pi}.\phi(\underline{s}) : \underline{s} \in \underline{S_{\widehat{\pi}}}(t)\}$ In conclusion, an output process $\widehat{\pi}$ of a virtual sensor $\widehat{s}$ is characterized by its observation function ϕ and its selection function ψ, i.e., $\widehat{\pi} = (\phi, \psi),\ \widehat{\pi} \in \widehat{s}.\widehat{\Pi}$.

As already mentioned, a virtual sensor may encompass multiple different output processes for observing a phenomenon property. Therefore, the set of results of all output processes $V_{\widehat{\Pi}}$ of a virtual sensor are represented by the union of the individual processes' outputs: $V_{\widehat{s}.\widehat{\Pi}}(t) = \bigcup_{\widehat{\pi} \in \widehat{s}.\widehat{\Pi}} V_{\widehat{\pi}}(t)$. Finally, a virtual sensor $\hat{s}$ for observing a property of type $\rho_i.\tau$ is characterized by the corresponding physical property type τ and its set of output processes $\widehat{\Pi}$. This virtual sensor provides all observations of a physical property (i.e., their values and associated regions) within the system space, such that the observer is able to merge the results according to the phenomenon state description vector $\vec{\rho}$.

4.3.2 Observer

The observer aggregates the results of virtual sensors $\widehat{S}$ such that an instance of the physical phenomenon $\vec{z}_{P_i}$ is created. It chooses a set of virtual sensors such that the state description vector $\vec{\rho}$ is covered. That is, for each property type $\rho_i.\tau$ of the state description vector the observer selects the corresponding virtual sensor $\widehat{s}$ that relates to the same property type $\widehat{s}.\tau$.

As described in the previous section, a virtual sensor provides multiple results. Hence, the observer performs a filtering operation on the results $V_{\widehat{s}.\widehat{\Pi}}$ of the virtual sensor's output processes. This encompasses choosing results $v_{\widehat{\pi}}$ of which the corresponding values $v_{\widehat{\pi}}.z_{\widehat{\pi}}$ satisfy the rule $\rho_i.r$ of the state description vector. The following equation summarizes this: $V_{\rho_i}(t) = \{(z_{\widehat{\pi}}(t), X_{\widehat{\pi}}(t)) \in V_{\widehat{s}.\widehat{\Pi}}(t) : \rho_i.r(z_{\widehat{\pi}}(t)) = 1,\ \widehat{s}.\tau = \rho_i.\tau\}$. As a phenomenon is present at locations where all of its properties are observed, the observer merges the results V_{ρ_i} for each element ρ_i in the state description vector. This merging operation $\mathcal{M}$ intersects the corresponding result regions such that individual cohesive regions remain for each of which all properties are present: $\left[(\vec{z}_{P_1}(t), X_{P_1}(t)) \quad \ldots \quad (\vec{z}_{P_m}(t), X_{P_m}(t))\right]^{\mathbf{T}} = \mathcal{M}(V_{\rho_1}(t), \ldots, V_{\rho_n}(t))$. The vectors $\vec{z}_{P_i}$ represent the values of these properties that stand for the observed state of a phenomenon P_i within its region X_{P_i}. Each of the resulting state vectors $\vec{z}_{P_i}$ forms an input for the behavioral function b of the described phenomenon in the application. In conclusion, the observer utilizes the available virtual sensors to realize the function g of the RTE presented in Section 4.2.

4.3.3 Virtual Actuators

A virtual actuator $\widehat{a}$ represents the capability to influence a physical phenomenon property of type τ. To achieve this, it may utilize different output processes $\widehat{\Pi}$, similar to a virtual sensor. For example, to increase the brightness within a room, electric lights may be turned on or window blinds may be lifted. An output process $\widehat{\pi}$ may utilize different sets of physical actuators that provide varying influences on a phenomenon property (e.g., turning on multiple lights or just one light to achieve varying levels of brightness).

The output process utilizes a selection function ψ to determine sets of devices that provide changes to a property of a phenomenon P_i located in a region X_{P_i}. Taking the phenomenon's locations into account is necessary since not all actuators are able to influence all regions within the system (e.g., due to their capabilities and physical contexts). Therefore, this selection function takes the capabilities, i.e., the classes ζ, of currently available actuators A, their effective output regions $X_{eff}(d)$, and their physical context regions $X_c(d)$ into account, similar to the selection function of output processes of virtual sensors. The selection function's result $\underline{A_{\widehat{\pi}}}$ corresponds to multiple physical actuator sets: $\underline{A_{\widehat{\pi}}}(t) = \widehat{\pi}.\psi(A(t), X_{P_i}),\ \underline{A_{\widehat{\pi}}}(t) \subseteq \mathcal{P}(A(t))$. Each of these sets is capable of providing changes to a property of a physical phenomenon that has the same type τ as the virtual actuator is related to.

As described in Section 4.2, the application requires a set of available influences on the described phenomenon properties from which it chooses the desired ones. Via a capability function β, each output process $\widehat{\pi}$ of a virtual actuator determines the set of property changes $\dot{z}_{\widehat{\pi},\underline{a}}$ it is able to provide via a device set $\underline{a}$, i.e., $\dot{z}_{\widehat{\pi},\underline{a}}(t) = \widehat{\pi}.\beta(\underline{a}),\ \underline{a} \in \underline{A_{\widehat{\pi}}}(t)$. The set of all available device changes per output process $\widehat{\pi}$ is denoted $\underline{\dot{Z}_{\widehat{\pi}}}$. This set is represented by the union of the influences $\underline{\dot{z}_{\widehat{\pi},\underline{a}}}$ made available by each set of suitable devices $\underline{a}$: $\underline{\dot{Z}_{\widehat{\pi}}}(t) = \bigcup_{\underline{a} \in \underline{A_{\widehat{\pi}}}} \underline{\dot{z}_{\widehat{\pi},\underline{a}}}(t)$. As a virtual actuator may encompass multiple output processes, the union of all available influences of the different output processes represents all available influences $\underline{\dot{z}_{\rho_i.\tau}}$ on a given phenomenon property of type $\rho_i.\tau$, i.e., $\underline{\dot{z}_{\rho_i.\tau}}(t) = \bigcup_{\widehat{\pi} \in \widehat{a}.\widehat{\Pi}} \underline{\dot{Z}_{\widehat{\pi}}}(t)$.

For each phenomenon property, the application chooses a desired influence $\dot{z}_{\widehat{\pi},\underline{a}}$ from the set of available influences $\underline{\dot{z}_{\rho_i.\tau}}$ (as depicted in Section 4.2).

Based on the application's choice, the corresponding output process $\widehat{\pi}$ utilizes an actuation function ϕ for determining the required input signals $\vec{u}_{\underline{a}}$ for the corresponding set of devices $\underline{a}$, i.e., $\vec{u}_{\underline{a}}(t) = \widehat{\pi}.\phi(\dot{z}_{\widehat{\pi},\underline{a}}(t))$, $\dot{z}_{\widehat{\pi},\underline{a}}(t) \in \dot{\underline{z}}_{\widehat{\pi},\underline{a}}(t)$. These input signals lead to the devices providing the desired influences on the physical world via their respective output processes (see Section 4.1).

In conclusion, an output process $\widehat{\pi}$ of a virtual actuator $\widehat{a}$ is characterized by its selection function ψ, its actuation function ϕ, and its capability function β such that $\widehat{\pi} = (\phi, \psi, \beta)$, $\widehat{\pi} \in \widehat{a}.\widehat{\Pi}$. A virtual actuator $\widehat{a}$ is characterized by its associated phenomenon property type τ and the set of its output processes $\widehat{\Pi}$.

The described virtual actuator model allows to determine the available influences on each phenomenon property described by the programmer. This is achieved by transparently selecting sufficient sets of physical actuators, determining their possible effects on the physical environment, and deducing required actuator input signals from a chosen influence on the physical phenomenon.

4.3.4 Controller

The controller utilizes virtual actuators to realize the functions $\underline{h}$ and $\bar{h}$ of the RTE (see Section 4.2). For an identified phenomenon instance P_j, it chooses the corresponding virtual actuators that relate to the same types $\rho_i.\tau$ as the phenomenon's properties. The controller creates a vector of available state influences $\dot{\underline{z}}_{P_j,ext}$ by gathering the results $\dot{\underline{z}}_{\rho_i.\tau}$ for each property from the corresponding virtual actuators: $\dot{\underline{z}}_{P_j,ext}(t) = \left[\dot{\underline{z}}_{\rho_1.\tau}(t) \quad \dots \quad \dot{\underline{z}}_{\rho_n.\tau}(t)\right]^{\mathbf{T}}$. The controller then invokes the application's behavioral function b (see Section 4.2) with the available state influences $\dot{\underline{z}}_{P_j,ext}$ and the observed state vector $\vec{z}_{P_j}$ as inputs. Therefore, the function $\underline{h}$ is realized.

As described in Section 4.2, the application's behavioral function b selects a desired influence for each property of the phenomenon such that it forms a desired state change vector $\dot{\vec{z}}_{P_j,ext}$. Each variable $\dot{z}_{P_j,ext,i}$ in this vector resembles an element from the set of available state influences $\dot{\underline{z}}_{\rho_i.\tau}$. Such an element refers to the influences an output process $\widehat{\pi}$ of a virtual actuator $\widehat{a}$ is able to provide, given a selection of physical actuators $\underline{a}$ as described in the previous section. This is summarized in the following equation: $\dot{z}_{P_j,ext,i}(t) = \dot{z}_{\widehat{\pi}_i,\underline{a}_i}(t), \dot{z}_{P_j,ext,i}(t) \in \dot{\underline{z}}_{\rho_i.\tau}(t)$, $\widehat{\pi}_i \in \widehat{a}.\widehat{\Pi}$. Based on the results of the behavioral function b (i.e., the application's chosen state influences $\dot{\vec{z}}_{P_j,ext}$), the controller distributes the individual state change variables $\dot{z}_{\widehat{\pi}_i,\underline{a}_i}$ to the corresponding virtual actuators. The following equation summarizes this for a phenomenon with a state description vector of length n: $\dot{z}_{\widehat{\pi}_i,\underline{a}_i}(t) \in \dot{\underline{z}}_{\rho_i.\tau}(t) \rightarrow \widehat{\pi}_i.\phi(\dot{z}_{\widehat{\pi}_i,\underline{a}_i}(t)) = \vec{u}_{\underline{a}_i}(t), \widehat{\pi}_i \in \widehat{a}_i.\widehat{\Pi}$, $\widehat{a}_i.\tau = \rho_i.\tau$, $i \in [1,n]$. These virtual actuators utilize their actuation functions ϕ to determine the required actuator input signals and therefore induce the desired change of state as described in the previous section (i.e., based on the chosen physical device $\underline{a}$ for the output process $\widehat{\pi}$): $\vec{u}_{P_j}(t) = \left[\vec{u}_{\underline{a}_1}(t) \quad \dots \quad \vec{u}_{\underline{a}_n}(t)\right]^{\mathbf{T}} = \left[\widehat{\pi}_1.\phi(\dot{z}_{\widehat{\pi}_1,\underline{a}_1}(t)) \quad \dots \quad \widehat{\pi}_n.\phi(\dot{z}_{\widehat{\pi}_n,\underline{a}_n}(t))\right]^{\mathbf{T}}$.

The resulting actuator input signals $\vec{u}_{\underline{a}_i}$ form the vector $\vec{u}_{P_j}$ for influencing the described phenomenon P_j. Therefore, the desired change of state is achieved and the function $\bar{h}$ is realized as described in Section 4.2. The controller removes the chosen actuators $\underline{a}_i$ from the currently available physical actuators for considerations on phenomena of lower priorities than the phenomenon P_j. Chosen input signals for phenomena of higher priorities overwrite signals for the same actuators that relate to lower-priority tasks.

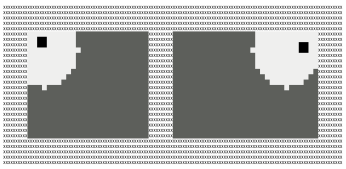
4.4 Initial Evaluation

We evaluated the presented concepts by simulating the running example of Section 3. The physical space was modeled as a two-dimensional field consisting of two distinct outward- and inward-blocking contexts (i.e., fabrication halls). Each location in this field encompasses information on its brightness and present object shapes. Human workers are placed within this field and change their location randomly. Figure 2 depicts the system space. The utilized physical devices consist of cameras and lights, capable of identifying object shapes and influencing the brightness of regions. Virtual devices encompass output processes for the localization of object shapes of interest and for achieving desired levels of brightness in given regions. The application model and RTE are realized as described in the previous sections.

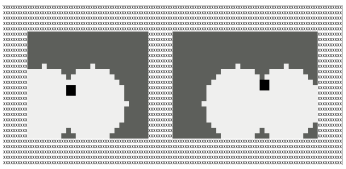
The simulation shows that the model's implementation operates as desired. We intend to further enhance the simulation and evaluate it with respect to the practicability and performance of our concepts. Based on the results we will refine our models and port them to physical systems. A detailed description of the simulation will be the subject of another publication.

5 CONCLUSION

In this work, we present a formal description for sensor and actuator virtualization concerning mobile and



(a) Screenshot of a simulation step in which one light is sufficient to achieve the desired brightness for a person.



(b) Screenshot of a simulation step in which two lights are required to achieve the desired brightness for a person.

Figure 2: Screenshots of different steps during the simulation of the running example (see Section 3) with two persons (black squares) in different contexts (enclosed by X) changing their positions. Illuminated areas are depicted in white, dark areas in dark gray.

heterogeneous CPS. It comprises a model of the physical environment that describes the capabilities of physical sensors and actuators with respect to their functionalities and the physical contexts they reside in. Additionally, our description encompasses the definition of physical phenomena that the CPS may observe or influence. Our approach includes an application model that enables the application developers to take a physical perspective to specify desired observations and influences of physical phenomena of interest. This allows the programmers to focus on the desired effect of the CPS, rather than having to directly interact with heterogeneous and mobile sets of physical devices. The RTE infers the required actions of the physical sensors and actuators according to the application's needs transparently (with respect to the application). We achieve this by introducing virtual sensors and actuators that represent the joint capabilities of possibly multiple physical devices that are mapped to observations and influences of physical phenomena of interest. Therefore, the application developers take a systemic view and I/O virtualization is introduced such that they do not have to explicitly interact with physical devices.

The realization of our model poses many challenges yet offers opportunities. From a theoretical point of view, a type system has to be introduced that precisely describes the operations the developer is able to perform on the provided data within the application. Additionally, models have to be created for the transparent coordination of multiple heterogeneous actuators. The implementation of such models for distributed CPS requires further considerations on performance, consistency, energy efficiency, and real-time capabilities.

REFERENCES

Beal, J. and Bachrach, J. (2006). Infrastructure for engineered emergence on sensor/actuator networks. In *IEEE Intelligent Systems*, pages 10–19.

Borcea, C., Intanagonwiwat, C., Kang, P., Kremer, U., and Iftode, L. (2004). Spatial programming using smart messages: Design and implementation. In *International Conference on Distributed Computing Systems*, pages 690–699.

Fernandez-Cortizas, M., Molina, M., Arias-Perez, P., Perez-Segui, R., Perez-Saura, D., and Campoy, P. (2023). Aerostack2: A software framework for developing multi-robot aerial systems. ArXiv preprint arXiv.2303.18237.

Laurenzi, A., Antonucci, D., Tsagarakis, N. G., and Muratore, L. (2023). The xbot2 real-time middleware for robotics. In *Robotics and Autonomous Systems*, page 104379.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. In *Science Robotics*, page eabm6074.

Ni, Y., Krember, U., Stere, A., and Iftode, L. (2005). Programming ad-hoc networks of mobile and resource-constrained devices. In *SIGPLAN*, pages 249–260.

Richter, M., Jakobs, C., Werner, T., and Werner, M. (2023). Using environmental contexts to model restrictions of sensor capabilities. In *17th International Conference on Mobile Ubiquitous Computing Systems, Services and Technologies*, pages 7–12.

Seiger, R., Seidl, C., Aßmann, U., and Schlegel, T. (2015). A capability-based framework for programming small domestic service robots. In *MORSE/VAO*, pages 49–54.

Suh, Y.-H., Lee, K.-W., and Cho, E.-S. (2013). A device abstraction framework for the robotic mediator collaborating with smart environments. In *International Conference on Computational Science and Engineering*, pages 460–467.

Tsiatsis, V., Gluhak, A., Bauge, T., Montagut, F., Bernat, J., Bauer, M., Villalonga, C., Barnaghi, P., and Krco, S. (2010). The sensei real world internet architecture. In *Future Internet Assembly*, pages 247–256.

Vicaire, P. A., Hoque, E., Xie, Z., and Stankovic, J. A. (2010a). Bundle: A group based programming abstraction for cyber physical systems. In *International Conference on Cyber-Physical Systems*, pages 32–41.

Vicaire, P. A., Xie, Z., Hoque, E., and Stankovic, J. A. (2010b). Physicalnet: A generic framework for managing and programming across pervasive computing networks. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–278.