Formal Methods in Designing Critical Cyber-Physical Systems

Mehrnoosh Askarpour[1], Carlo Ghezzi[1], Dino Mandrioli[1], Matteo Rossi[1], and Christos Tsigkanos[2]

[1] Politecnico di Milano, DEIB, Milan, Italy
name.surname@polimi.it
[2] Vienna University of Technology, Vienna, Austria
name.surname@tuwien.ac.at

Abstract. Cyber-Physical Systems (CPS) are increasingly applied in critical contexts, where they have to support safe and secure operations, often subject to stringent timing requirements. Typical examples are scenarios involving automated living or working spaces in which humans operate, or human-robot collaborations (HRC) in modern manufacturing. Formal methods have been traditionally investigated to support modeling and verification of critical systems. In this paper, we review some of the main new challenges arising in the application of formal methods to modeling and verification of CPS. We do that by presenting two case studies (emergency response in a smart city and a smart manufacturing system), reflecting past work of the authors, from which some general lessons are distilled.

Keywords: Cyber-Physical Systems (CPS)· Formal Model· Formal Verification· Model-based Design

1 Introduction

The revolutionary advancements of embedded computing have led to a generation of systems that integrate computing and physical processes, called cyber-physical systems (CPS) [23,2]. Such systems incorporate functions of sensing, actuation, and control while making decisions in a predictive or adaptive manner. This manifests in various novel fields such as the Internet of Things (IoT) [3]. The use of CPSs is growing every day with the developments of new application areas. For example, CPSs enable the creation of smart spaces [29,26], i.e., spatial environments including both cyber and physical elements and supporting new kinds of advanced functionalities. A particular case of smart spaces are smart factories [33], where computational and communication features are embedded in a manufacturing workspace to combine the flexibility of humans with the efficiency of machines and to allow for collaboration between them in a safe way [25]. To design such new kinds of complex systems, it is crucial to analyze, specify, and then verify their expected properties. Often properties are classified in functional vs non-functional ones, where the former capture the

expected results of the system, whereas the latter correspond to its complementary properties—such as space, time, safety, security, fault tolerance, continuous adaptation, communication and process time, energy, or cost—and are no less relevant than the functional ones.

Model-based techniques could considerably simplify the design of such complex systems, and make the analysis of all of their required properties precise and rigorous. Although the separation of cyber and physical concerns in the modeling and design of CPSs could be beneficial for tractability, it complicates the assessment of the impacts and tradeoffs of the two domains [22]. The interplay between cyber and physical elements raises new challenges, and their effective orchestration requires semantic models that reflect properties of interest in both of them [19].

Formal languages should be defined to support specification both of a formal model of a CPS—e.g., a smart space—and of the properties it is expected to satisfy. These models could be input to automated formal verification tools to enable analysis and validation during design. Furthermore, the physical aspect of CPSs brings more uncertainty and dynamism into the picture w.r.t traditional embedded systems, due to the *runtime* physical intercommunication with the world (e.g., human-robot interaction, sensing or actuating on the elements of the environment). Thus, formal methods should also be brought to runtime, to support runtime verification and possible automatic adaptations to detected changes.

In this paper, we report the results of our analysis of the state of the art concerning the main issues regarding the modeling of CPSs and discuss the value of formal methods in resolving them. The paper reviews three issues that we found highly critical and then describes two case-studies –a smart city and a smart factory– designed and verified by formal modeling and validation techniques.

The rest of this paper is structured as follows: Section 2 argues about the most important challenges in the modeling of CPSs and the use of formal methods to address them; Section 3 reports two case-studies designed and verified in a formal manner; finally, Section 4 concludes.

2 Key Factors in the Design of CPSs

This section introduces a set of common, critical issues that have been raised in the literature concerning the design of CPSs.

2.1 Space and Time

From a software engineering perspective, cyber-physical systems live within a dynamic spatial environment populated with devices, human agents, changing context and/or localized resources. This can be abstracted into a cyber-physical space (CPSp), a structure indicating a spatial environment comprised of both computational and communication elements, which are interrelated and form some composite topological structure [29]. Such cyber-physical spaces are much

more dynamic than traditional—physical—spatial environments used to be. Humans or devices moving around connecting and disconnecting from wireless networks are an example of entities dynamically performing actions while operating in a composite CPSp. Such dynamics have to be considered in the design of systems operating within spatial environments. Moreover, as for any other software-intensive system, maintaining a CPSp which operates in a dynamic environment is faced with the manifold challenges of software system evolution [13,32] and demands for operational management to observe a constantly changing space and potentially react to environmental changes.

The physical environment in which CPSs operate or where CPSps are realized, is perceived not only across space, but also over time. Hence, CPSs exhibit spatio-temporal features and their correct behavior is defined in both space and time. A formal representation of a CPS should predicate upon the flow of events along time, while reflecting spatial characteristics such as the distribution of systems in space [6], including positions or distances of different components. Such spatial characteristics also affect the timing of events within the system and the overall execution workflow, as only particular spatial distributions of objects would lead to the correct and safe execution of the business logic. Moreover, even though the resulting models should be a reflection of the physical reality, they should also suit formal verification. Hence, these models are usually an approximation of the reality resulting by appropriately abstracting the temporal [11] and spatial domains in which the system lives. In other words, the formal models of CPSs, like those of other types of systems, trade precision and exhaustiveness for simplicity and tractability and accept some level of abstraction.

Let us illustrate through an example the importance of capturing both space and time requirements in CPSs. Consider a manufacturing workspace in which humans and robots collaborate without any fences or physical segregation between them. A formal model of such a system needs to capture different temporal requirements of the collaboration (e.g., what the expected response time is, or what is the sequencing of jobs the robot should execute before the operator) and also spatial characteristics of the workspace (e.g., which mobility paths are more frequent for robots, where the exact place for execution of each job is, which areas are more prone to more frequent and dangerous contacts between humans and robots). It is important for robots and humans to perform the right action, at the right time and in the right place. For example, assume that a human and an industrial robotic arm with a screwdriver end-effector are expected to perform a collaborative pick-and-place task (e.g., picking workpieces with different shapes from a bin, then place and screwdrive them on a pallet). The correct way of performing this task is the following: "first the operator picks the workpieces one by one and places them on the pallet, then she removes her hand from the pallet; only then the robot end-effector moves above the pallet and starts to screwdrive the pieces". If the operator violates this instruction and, for example, tries to slightly move the workpiece while the robot is screwdriving it, then the execution may be interrupted—hence prolonged—or, worse, the operator could get hurt.

The interplay between space and time and its relevance to the satisfaction of requirements is exacerbated for novel types of pervasive systems, technologies and paradigms such as the Internet of Things (IoT), which often feature physically distributed entities roaming inside the physical space [30], exhibiting collective behaviors. Spatially-distributed IoT systems live within a dynamic spatial environment populated with devices, a changing context and/or localized resources. This spatial environment is often only partially known—or even unknown—at design time, which creates the need for suitable reasoning facilities and analyzable models used to observe, evaluate and react to a constantly changing space [35]. Frequently, these activities must be performed during the system operation, when analysis techniques working at runtime ensure that possible changes occurring due to the evolving spatial distribution and context—for example due to actions performed by active agents, or by the external environment—do not lead to violations of requirements [28]. This can be achieved through an autonomic, self-adaptive approach such as a MAPE loop [18]: (M)onitoring the spatial environment for changes, (A)nalyzing possible requirement violations, (P)lanning necessary countermeasures (e.g., moving a device from one point in space to another) and then (E)xecuting such actions and updating the shared model of space for the next loop.

2.2 Human-Robot Interaction

A distinguishing feature of many smart space applications is the presence of interactions between humans and robots. We can broadly classify the types of robots involved in these applications in the following categories:

- **Robots with interface devices** are such that the operator usually has no physical contact with the robot and the communication occurs via interface devices. They are used in a variety of domains such as healthcare, manufacturing, disaster management. Examples are medical and surgeon robots, large manufacturing robots and earthquake rescue robots.
- **Service provider or domestic robots** are usually employed as caregivers for elderlies or people with physical disabilities. Interaction with this type of robots occurs via interfaces, but it also features some level of physical contact with human operators (i.e., the care receivers). In the applications where these types of robots are used, the output of the robot is not continuously dependent on human inputs, and once a command is received from the interface device, the robot proceeds with its execution. In these scenarios humans are mostly passive receivers of services.
- **Collaborative robots** should attain a predefined goal by working in collaboration with humans. The specified objective—i.e., a job—is typically divided into atomic parts—i.e., actions— that must be carried out either by humans, or by robots, or by both of them concurrently. In these scenarios robots coordinate their actions with those of the humans—e.g., the operator must place a workpiece in position x before the robot can screwdrive it there.

These are general types of robots and could potentially be applied in very different areas, from disaster response to entertainment.

In general, the presence of humans (either as physical participants, or as command triggers)—who are, by their own nature, unpredictable agents—raises significant modeling issues when one wants to guarantee a certain level of safety and reliability for the application. To capture the unpredictable nature of human actions, stochastic models could be defined to describe the probability with which a certain behavior is taken, thus allowing designers to focus on the most probable ones. Building meaningful stochastic models, however, requires huge and reliable log data concerning human actions—and in particular human interactions with robots—to identify suitable probabilistic distributions (e.g., how probable it is for the operator to make an error and perform an action earlier or later than when it needs to be done). Unfortunately, such data logs are usually not available. To overcome this problem, nondeterministic modeling approaches can be used, which render unpredictability by describing alternative behaviors that are chosen in a nonobservable manner. Formal models could, for example, capture reasonably foreseeable human behaviors, for different types of human operators (experienced user vs novice user, attentive vs absent-minded, etc.). Nondeterministic and stochastic models could also be combined; for example, nondeterminism could be used to describe the choices that an operator can make based on her level of fatigue, but each fatigue level could have a different probability.

2.3 Managing Uncertainty at Runtime through Self-Adaptation

The difficulty of requirement validation for CPSs is exacerbated by the fact that they include both computational and physical aspects, they are susceptible to emergent behaviors, and their operational environment is often only partially known—or even unknown—at design time. Therefore, requirement analysis—preferably through formal verification—is a key activity in the design of CPSs. Requirement analysis consists in evaluating whether the system (as deployed in some environment) satisfies some intended behavior. However, the extent of the assurances obtained through the analysis depends on whether they address concerns that arise at design time or at runtime.

A consequence of relying only on design time verification is that the quality of provided guarantees depends strongly on the quality of the generated model. If the people who brainstormed to build the model have left out even only one possible situation, serious problems could occur during system operation (e.g., issues arising from people forgetting to take back their cards from an ATM). Consider, for example, a smart manufacturing facility where both humans and robots operate. In addition to the sources of uncertainty that are known at design time (e.g., human errors, malfunctioning sensors and actuators), unexpected events could occur during system operation, which had not been foreseen at design time. Examples of such events could be the unplanned entrance of another human operator in the workcell, or the unloading of workpieces to be grabbed by a robot that are geometrically unknown for the robot gripper. Even sources

of uncertainty that are foreseeable at design time can be difficult to manage at runtime, as they could generate special sub-cases that have not been analyzed during design. For example, the fact that humans might make errors when interacting with a robot is quite expectable, but it is very difficult to consider all possible such errors and their potential critical consequences.

The above example shows that satisfaction of certain requirements cannot always be guaranteed at design time. Instead, evaluation must be deferred at runtime, and subsequently their satisfaction must be ensured by adaptively generating counteractions that can prevent the system from violating requirements [31]. These counteractions, in turn, may threaten the satisfaction of system requirements. Uncertainty and its attributes has been investigated in the past [21,10], and classified [21] by (1) the place where it manifests, (2) its level, and (3) its nature—i.e., whether it originates from imperfect knowledge or from variability. Research on self-adaptive systems has long tackled managing uncertainty at runtime, considering both functional and non-functional requirements [4].

3 Case-studies

In this section we report on two exemplars of critical CPSs where formal methods prove to be highly useful. The first one is a disaster scenario within a smart city, highlighting both design time reasoning of a space-intensive CPS as well as its runtime verification. The second is a smart factory, highlighting the validation of domain-specific requirements concerning the human operator's physical safety during the interaction with a robot system. Different formalisms and modeling approaches are adopted to enable reasoning on the two case studies, showing the potential that formal methods can bring to the design and analysis of complex CPSs.

Let us remark that the main concerns in the two case studies are different. The first example is more oriented towards the verification of general concepts (safety, reliability, integrity and confidentiality) with stronger focus on spatial system aspects and their modeling; the second one verifies domain-specific requirements (physical safety of human while interacting with the robot system) with a stronger focus on temporal aspects.

3.1 Case study one: Reasoning on Space-Intensive CPS

In this section, we introduce a case study concerning the emergency response in a smart city as an illustrative scenario of a space-intensive CPS, reasoning upon which is enabled by its consideration as a cyber-physical space (CPSp). We describe the scenario in a succinct manner; the interested reader can refer to [29] for a complete treatment. The scenario presented is a generalized case which can be instantiated for a variety of spatio-temporal reasoning cases. We begin with a brief description of the static structure of the cyber-physical space and then we consider its dynamics—i.e., how the space may change over time. Subsequently, we introduce two characteristic analysis scenarios exposing typical

design challenges that are relevant for design and operation. We suggest that satisfaction of critical requirements arising from these scenarios can be either checked at design time, or at runtime.

Autonomous Unmanned Aerial Vehicles (UAVs) can be used as radio relay platforms in environments characterized by poor connectivity. These environments can be regions where no global connectivity exists, e.g. due to a disaster or even absence of line of sight between ground transmitters and receivers. We consider a setting of UAV-carried communication infrastructure [34] in a disaster scenario for smart city applications such as emergency response. The setting we present, including the model and its dynamics, is a generalized case [9] which can be concretized for a variety of urban warfare, search and rescue, homeland security or surveillance scenarios where autonomous UAVs operate in a space-dependent environment and global system properties need to be formally verified.

Emergency Response in a Smart City. Communication is disabled in a city due to a disaster; search and rescue must be performed. Parts of the city may be unsafe, and victims may be stranded in various locations. Autonomous UAVs are dispatched to locate and provide communication infrastructure to victims, leading them to safety. UAVs move in the city environment in specific ways, by flying over buildings. UAVs carry short-range antennas, and victims are able to connect when they are in the vicinity. If a UAV is close to a victim, it can lead her to a safe zone. A safe zone is some part of the city which can lead to a hospital. To utilize our approach, the designer specifies the model, the ways UAVs can move and desired properties of the system, specification steps illustrated in the following.

Modeling Space and its Dynamics. In general, graphs are a natural way to model the topology of a CPSp, such as the urban scenario at hand. The basic intuition is that entities are represented by nodes, while relations between entities are represented by edges. We distinguish two fundamental kinds of relations between entities to which we refer to as *containment* and *linking*. Containment signifies that an entity is located within another, while linking expresses the fact that two entities are connected in some way. In Figure 1, the topological structure of a city is presented, where buildings, roads and city blocks form a city. Various such entities may be connected, signifying that one can physically move from e.g. a building to an adjacent one. Such a model may enjoy formal semantics.

Bigraphs [20] are an emerging formalism for structures in ubiquitous computing, dealing with both containment and linking among entities and thus fit our intuition of modelling the topology of a cyber-physical environment. We use the basic notion of a *bigraph* which consists of two superimposed yet orthogonal graphs: a *place graph* is a forest, a set of trees defined over a set of nodes, and a *link graph* is a hypergraph over the same set of nodes, where edges between nodes can cross locality boundaries. Nodes are typed, and the node types are called *controls* in the bigraphical terminology.

We abstain from providing details of the formalism and instead rely on intuition; the interested reader can refer to the vast body of literature on the topic for complete definitions and proofs the bigraphical theory [20]. Bigraphs can be described in algebraic terms according to Formulae 1a-1e. Basically, nodes are written in terms of their controls, i.e., names that define a node's type, such as P, Q, and U. The hierarchical structure of nodes through containment relationships is expressed according to Formula 1a, while the notation in Formula 1b is used to indicate that two nodes are placed at the same hierarchical level. Bigraphs form rooted hierarchies; in Formula 1c, W and R indicate different roots. Bigraphs can contain *sites*, a special kind of node that denotes a placeholder, indicating the presence of unspecified nodes. A node may contain any number of sites, which are simply indexed in the context of their defining bigraph, as expressed in Formula 1d. Second, connections of an edge with its node are treated as separate elements of a bigraph, referred to as *ports*. Port names appear in the algebraic notation; in Formula 1e, the node of control K has port names in w. These port names are used to identify nodes, which may be omitted if a single instance node of a given type exists in the bigraph, and to express the linking structure: ports with the same name are connected forming a hyper-edge in the link graph. Port names prefixed by '@' are variables ranging over the names of a bigraph.

$\mathsf{P}.\mathsf{Q}$	*Nesting* (P *contains* Q)	(1a)
$\mathsf{P} \mid \mathsf{Q}$	*Juxtaposition of nodes*	(1b)
$\mathsf{W} \parallel \mathsf{R}$	*Juxtaposition of bigraphs*	(1c)
$-_{\mathsf{i}}$	*Site numbered* i	(1d)
K_{w}	*Node with control* K *having ports* w	(1e)

In practice, we can obtain a bigraphical model of space for our smart city case study from a domain model; this occurs in two steps. To obtain the basic topological structure of a city, we automatically extract a bigraph from city models described in CityGML [14], a widely used XML-based standard for the exchange of city models, widely used within the architectural informatics discipline. Subsequently, further entities of interest such as UAVs and disaster victims are placed in that model. A conceptual representation of the topological structure extracted from a CityGML model with 20 buildings is illustrated in Figure 1. A 2D projection of the roads and buildings is shown in light grey in the background, while the conceptual bigraphical structure is shown in the foreground. The bigraph exposes the following placing structure: A City node serves as root of the extracted bigraph. It contains nodes of type Road which in turn contain nodes of type RoadSegment and Crossroad, a road segment representing the part of a road between two crossroads. Moreover, a City node contains nodes of type Block, a block representing the area surrounded by road segments. Blocks may contain an arbitrary number of Building nodes, each one representing a building. Auxiliary nodes (e.g. for City and Road) are not shown in Figure 1 for sake of readability. Other entities are present in the city as well, such as hospitals, airports, etc. As for the linking structure of the extracted bigraph, each building is connected to the building next to it (represented by blue links in Figure 1), and to a block's surrounding road segment if it is located in the respective block

boundary (represented by green links). Moreover, road segments are linked to the crossroads being connected by that road segment (represented by red links). An equivalent, partial algebraic representation of the city space is found in Formula 3; the formula shows how two buildings, Bld_2 and Bld_4 (found within the same block Blk_1), are connected through a link C_4, signifying that UAV_3 may go from one to another. A *Hospital* is also contained in the city block (not shown in Figure 1).Other entities in various parts of the model are abstracted away in the formula representation using the formalism's sites facility.

$$\begin{aligned} &\mathsf{City}.(\mathsf{Blk}_1.(\mathsf{Bld}_2.(\mathsf{UAV}_3.(-_{11}) \mid \mathsf{Bg2Bg}.(\mathsf{C}_4 \mid -_1) \mid -_5) \\ &\mid \mathsf{Hospital} \mid \mathsf{Bld}_4.(\mathsf{Bg2Bg}.(\mathsf{C}_4 \mid -_4) \mid -_3) \mid -_2) \mid -_9 \mid -_{10}). \end{aligned} \tag{2}$$

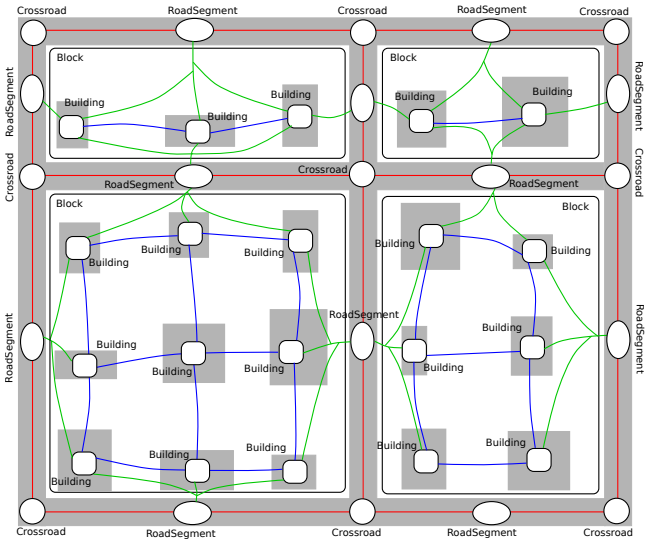

Fig. 1: Bigraphical structure extracted from a city model.

Space is rarely static, thus a formalism for modeling evolving space-intensice systems should also capture system *dynamics* to enable reasoning about the effects of changes in space. Bigraphical Reactive Systems (BRS) [20] extend bigraphs with well defined semantics of dynamic behavior expressed as a set of rules. BRS essentially allow describing possible ways in which the structure of the space can evolve through the application of transformation rules which selectively rewrite parts of a bigraph; they are called *reaction* rules. Reaction rules have the general form of $\mathsf{R} \rightarrow \mathsf{R}'$, where R is called the *redex* and R' is called the *reactum*; both the redex and reactum are bigraphs. If an occurrence of a redex can be found in a host bigraph, it may be replaced by the reactum, in a fashion similar to graph rewriting [8]. Redex and reactum can be considered

as *patterns*, which are parametric; they describe some structure that can be transformed into another, which may not be concretely specified.

To this end, we can model the changes inherent in the disaster scenario using a BRS specification. For reasons of simplicity, we consider one such type of dynamics specification; how UAVs may move from a building to another. In Formula 3, a parametric reaction rule captures how a UAV moves from a building ($\mathsf{Bld}_{@\mathsf{a}}$) to another connected one (i.e., through a connection $\mathsf{C}_{@\mathsf{b}}$ which has the same port name). Essentially, the CPSp model is transformed, to record that the UAV is now found inside the $\mathsf{Bld}_{@\mathsf{b}}$. Note how the reaction is parametric; presence of other UAVs, buildings or other entities is not described, but merely that the UAV moves inside the structure specified. Similarly to this reaction rule, we may additionally consider e.g. that victims located by UAVs move with them until a safe zone is reached.

$$\begin{aligned} &\mathsf{Blk}_{@\mathsf{w}}.(\mathsf{Bld}_{@\mathsf{a}}.(\mathsf{UAV}_{@\mathsf{UAVid}}.(-_{11}) \mid \mathsf{Bg2Bg}.(\mathsf{C}_{@\mathsf{b}} \mid -_1) \mid -_5) \\ &\quad \mid \mathsf{Bld}_{@\mathsf{b}}.(\mathsf{Bg2Bg}.(\mathsf{C}_{@\mathsf{a}} \mid -_4) \mid -_3) \mid -_2) \mid -_9 \rightarrow \\ &\quad \mathsf{Blk}_{@\mathsf{w}}.(\mathsf{Bld}_{@\mathsf{a}}.(\mathsf{Bg2Bg}.(\mathsf{C}_{@\mathsf{b}} \mid -_1) \mid -_5) \\ &\mid \mathsf{Bld}_{@\mathsf{b}}.(\mathsf{UAV}_{@\mathsf{UAVid}}.(-_{11}) \mid \mathsf{Bg2Bg}.(\mathsf{C}_{@\mathsf{a}} \mid -_4) \mid -_3) \mid -_2) \mid -_9. \end{aligned} \tag{3}$$

Analysis Scenarios and Verification. We consider two different analysis scenarios; the first aims at early requirements validation, as typically performed at design time. The second highlights validation at runtime, where the underlying system model is only known while in operation.

Scenario A: Verification of System Requirements. While bigraphs and bigraphical reaction rules are adequate for describing the topology of a CPSp and its inherent dynamics, a quality evaluation model to support systematic reasoning on the behaviour of the changing system is required. We assume that a CPSp is specified by a BRS as discussed previously. To enable automated reasoning, this specification can be transformed into an equivalent transition system generally known as a (Doubly) *Labelled Transition System* (dLTS) [7]. States of this transition system describe bigraphical configurations of the CPSp, while transitions describe how the configuration of the system can change by moving from one state to its successors. Interpreting a BRS specification as a dLTS entails describing its possible evolution based on the application of reaction rules. Labelled transition systems are amenable to formal verification via explicit-state model checking. Model checking performs an exhaustive analysis of the state space to check the validity of a property. We abstain from describing the mechanisms behind this, and instead illustrate a characteristic case; the interested reader can refer to [29] for a complete treatment.

We consider the setting where victims and UAVs are positioned in various parts of the city; the initial state of the system is thus known. Recall that there is a hospital in the city, and that victims are considered safe if they are in the hospital. UAVs roam inside the city, and if they locate a victim, they

lead it to safety. Normally, UAVs follow some path planning strategy [24]; from all possible movements of a UAV at any point, a strategy selects the optimal, based on the strategy and local environmental conditions. Moreover, interesting problems arise with target search and surveillance scenarios, which can lead to complex controller algorithms [9]. We are not concerned with the design of a controller here, but with verifying properties of the system which concern *any* decisions that the system of UAVs may take while operating within a city environment. Behaviors that may violate a global property of the system must be investigated, so every possible system behavior must be verified, possibly with an overall goal of using violating sequences to learn (or debug) a controller strategy. We consider a generic global requirement of the system, which states that if victims exist in the city, eventually all victims are found inside the hospital (i.e., no victims are located in other buildings). An LTL property (with the usual semantics) encoding the requirement is found in Formula 4, utilizing a parametric bigraphical pattern to express that no victim is eventually found in a building. Such a property can be used to validate the domain modeling, by verifying that the model and dynamics specified indeed lead to a valid system.

$$\Box(\mathsf{Victim} \rightarrow \Diamond\neg\mathsf{Building}_{?}.(\mathsf{Victim} \mid -_0)). \tag{4}$$

Scenario B: Spatial Verification at Runtime. In this scenario, we assume that UAVs are deployed in a spatial environment that is unknown at design time. The spatial model is instead built and updated at runtime, for example through a monitoring infrastructure in place [30]. Furthermore, we consider the following property, specifying that "all disaster victims are located in places in the city so that they can reach the hospital through road segments or crossroads". This is an example of a property of interest that cannot be expressed in LTL, since it does not predicate about the temporal evolution, but about certain relations in space – its topology. Hereafter, we illustrate the use of a spatial logic by which we can capture and verify the property at hand. We abstain from describing precisely semantics and verification procedures, and instead illustrate an exemplar case; the interested reader can refer to [29] for a complete treatment.

The spatial reasoning approach is based on SLCS [6], an extension of the topological semantics of modal logics to closure spaces, a closure space being a generalization of a standard *topological* space [12]. SLCS is a spatial logic for closure spaces which serves as the basis for our approach to complex reasoning over topological properties of a CPSp. A logical property will be evaluated accordingly upon updated models of the CPSp obtained at runtime, assuming no knowledge about the structure of the model at design time, beyond the actual property specification. A spatial formula in our case, consists of propositions representing bigraphical patterns along with SLCS operators. The logic features boolean operators, a "one step" modality turning closure into a logical operator, and a surrounds operator. Given that p is drawn from a set of bigraph patterns, the syntax of SLCS is defined by the following grammar:

$$\phi ::= \mathsf{p} \mid \top \mid \neg\phi \mid \phi \wedge \psi \mid \mathsf{C}\ \phi \mid \phi\ \mathsf{S}\ \psi. \tag{5}$$

In Formula 5, $\top$ denotes true, $\neg$ is negation, $\wedge$ is conjunction, C is the closure operator, and S is the spatial surrounds operator. When used for the sake of spatial model checking, SLCS formulae are evaluated upon bigraphical closure models [29]. While the elementary syntax presented in Formula 5 features the two fundamental spatial operators of closure and surrounds, a set of more complex operators may be derived from them. In [5], for instance, complex operators reflecting the notions of nearness and reachability have been derived. In particular, the so-called "reach through" operator is defined as $\phi\ \Re(\psi)\ \zeta$. Informally, it is satisfied for a point x, if x satisfies ϕ and there is a sequence of points starting from x, all satisfying ψ, reaching a target point satisfying ζ.

Formula 6 below formally encodes a property which needs to be verified upon a model monitored at runtime. Note that there is no information encoded on *how* a victim should reach the hospital, and the specification is able to capture every possible instance of a reachability realization through crossroads and road segments that may appear on a model.

$$\mathsf{Victim}\ \Re(\mathsf{RoadSegment} \vee \mathsf{Crossroad})\ \mathsf{Hospital}.(-_1)). \tag{6}$$

To support runtime verification of the CPSp in our operational scenario, properties like the one presented in Formula 6 can be evaluated whenever the monitoring indicates a change in the CPSp's toplogy, which is reflected into the bigraphical closure model. In the simplest case, an alarm may be generated if a critical property is violated. More advanced systems could be self-adaptive, counteracting property violations by triggering measures that ensure that requirements are satisfied. While the specification of such systems is beyond the scope of this paper, related research has proposed a number of such strategies [27]. Although self-adaptation has been largely studied for temporal properties, we believe that the fundamentals may be adopted for the spatial domain as well.

3.2 Case study two: Reasoning on Temporal modeling of CPS

In this scenario, we have analyzed a mobile robot unit which autonomously relocates in the layout shown in Fig. 2. This robot system is configured as a combination of a driverless truck (i.e., AGV) and a manipulator, which mainly moves between three assembly stations—(1), (2) and (3)—and a sensor-based inspection station (4), as shown in Figure 2(a). The robot unit can be manually relocated by operators around its predefined positions. The robot unit can travel and access the whole workspace (the blue area in Figure 2(b)), including a load/unload area for raw materials and finished parts. Two human operators (OP_1 and OP_2) are employed in the application. OP_1 is mostly present in stations (1) and (2), while OP_2 works mainly in (3) or executes auxiliary manual tasks on the workbench in (4). Both operators can freely hold and resume their tasks, swap posts, or join one another in some area. The main robot-assisted intended tasks are: pallet assembly at stations (1) and (2), including bin-picking

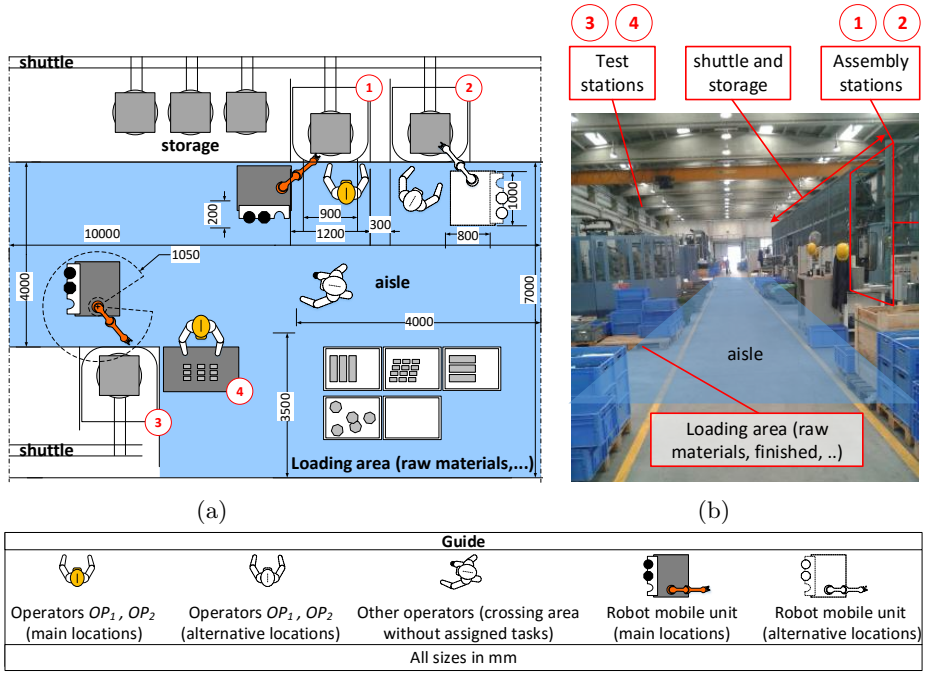

Fig. 2: Sketch of a cyber-physical space of a smart factory: (a) precise workcell depiction, (b) actual layout.

from a local storage carried by the mobile unit; pallet disassembly (reversal of assembly) at (1) and (2), including bin-dumping; pallet inspection at station (3); lead-through programming of assembly, disassembly, and inspection tasks (trajectories, parameters, etc.) at stations (1), (2) and (3); material handling on load/unload areas. Other manual tasks by OP_1 and OP_2 include manual loading of parts/boxes; (additional) visual inspection of pallet at stations (1), (2) and (3); manual assembly/disassembly of pallet at stations (1) and (2); manual measurements of parts at station (4); cleaning pallets at stations (1) and (2); kitting of tools and parts at stations (1), (2) and (3); general supervision at stations (1), (2) and (3). The generated formal model of the described system replicates *all* combinations of robot/manual task assignments (e.g., robot holds and OP_1 screw-drives jigs and vice versa, switching tasks on the fly, quitting a manual task and assigning the robot to proceed autonomously). Frequently, robot base and operators move side-to-side across the central aisle, or other operators transit along the aisle because the target area is part of a larger plant and access to it is not restricted (Fig. 2(b)).

Table 1: List of derived TRIO operators; ϕ, ψ denote propositions, and v is a variable and d is a constant value.

TRIO Operator	Definition	Meaning
$\mathsf{Past}(\phi, d)$	$d > 0 \wedge \mathsf{Dist}(\phi, -d)$	ϕ occurred d time units in the past
$\mathsf{Futr}(\phi, d)$	$d > 0 \wedge \mathsf{Dist}(\phi, d)$	ϕ occurs d time units in the future
$\mathsf{Alw}(\phi)$	$\forall t(\mathsf{Dist}(\phi, t))$	ϕ always holds
$\mathsf{Som}(\phi)$	$\exists t(\mathsf{Dist}(\phi, t))$	ϕ occurs sometimes
$\mathsf{Until}(\phi, \psi)$	$\exists t(\mathsf{Futr}(\psi, t) \wedge (\forall t'(0 < t' < t) \Rightarrow \mathsf{Dist}(\phi, t')))$	ψ will eventually occur and ϕ will hold till then
$\mathsf{Until_w}(\phi, \psi)$	$\mathsf{Until}(\phi, \psi) \vee \mathsf{Alw}(\phi)$	weak until: ψ may never occur in the future
$\mathsf{WithinF}(\phi, d)$	$\exists t(0 < t < d \wedge \mathsf{Dist}(\phi, t))$	ϕ will occur within d time units

Modeling Time and its dynamics. The model includes a descretized replica of human ({head, chest, leg, arm, fingers}) and robot $\{R_1, R_2, R_{ee}, R_{base}\}$. This example, unlike the previous one, focuses more strongly on time. This preference is driven by the overall goal of the project, which was centered on human physical safety analysis. As described in Section 2.2, different timings of human and robots actions could lead to different situations which could be harmful to human. Harmful situations could also be of different intensities and need to be evaluated differently. For example if human and robot have a contact in (1), the harmfulness of the contact could differ if the robot is already there and then human enters (1) and hits the robot or viceversa.

In order to model these temporal configurations we have used TRIO, a logical language which assumes an underlying linear temporal structure and features a quantitative notion of time [11]. TRIO formulae are built out of the usual first-order connectives, operators, and quantifiers, as well as a single basic modal operator, called Dist, that relates the *current time*, which is left implicit in the formula, to another time instant: given a time-dependent formula ϕ (i.e., a term representing a mapping from the time domain to truth values) and a (arithmetic) term t indicating a time distance (either positive or negative), formula $\mathsf{Dist}(\phi, t)$ specifies that ϕ holds at a time instant at a distance of exactly t time units from the current one. While TRIO can exploit both discrete and dense sets as time domains, in this work we assume the standard model of the nonnegative integers N as discrete time domain. For convenience in the writing of specification formulae, TRIO defines a number of *derived* temporal operators from the basic Dist, through propositional composition and first-order logic quantification. Table 1 defines some of the most significant ones.

Yet, modeling the space is very important because of the placement of sensors and endeffectors of the robot. The workspace is discretized in 23 regions with different characteristics and not all areas have static properties. The robot is

constantly moving around and the areas mostly situated on the aisle could have different characteristic from time to time. In order to resemble the human reasoning about spatial properties and construct a 3D model of the workspace, each region is modeled in three layers: lower, middle and upper sections. The mobile base of the robot is always allowed in the lower layer, while the manipulator arm could move in the middle and upper layers.

For example in order to capture the contact in which robot hits the operator, predicate $\mathsf{ArrivedBefore}_{ijk}$ is introduced which states that the human part $i \in$ {head, chest, leg, arm, fingers} hits robot part $j \in \{R_1, R_2, R_{ee}, R_{base}\}$ in the layout section $k \in 1...23$ because it arrived at k earlier than j. This formula is only an example of the interplay between time and space, as explained in Section 2.1. Examples of other formulae are presented in Table 2[3]. The relative values of force

Table 2: Selected formulae of the second case study.

The parts of discretized operator, robot and layout are shown by O_i, R_j, L_k.

1.$\forall O_i \in$ { head, chest, leg, arm, fingers}
2.$\forall R_j \in \{R_1, R_2, R_{ee}, R_{base}\}$
3.$\forall O_i . \exists! L_k (p_i = L_k)$
4.$\forall R_j . \exists! L_k (p_j = L_k)$
5.$\mathsf{Sep}_{ij} \in \{\texttt{close}, \texttt{mid}, \texttt{far}\}$
  $a.\mathsf{Sep}_{ij} = \texttt{close} \leftrightarrow \exists L_k (p_i = p_j = L_k)$
  $b.\mathsf{Sep}_{ij} = \texttt{mid} \leftrightarrow \mathsf{Adj}(p_i, p_j)$
  $c.\mathsf{Sep}_{ij} = \texttt{far} \leftrightarrow \mathsf{Sep}_{ij} \neq (\texttt{mid}|\texttt{close})$
6.$\mathsf{InSameL}_{ijk} \Leftrightarrow \mathsf{Sep}_{ij} = \texttt{close}$
7.$\mathsf{ArrivedBefore}_{ijk} \Leftrightarrow \mathsf{InSameL}_{ijk} \wedge \mathsf{Past}(p_i = L_k, 1) \wedge \mathsf{Past}(p_j \neq L_k, 1)$
8.$\mathsf{Reach}_{ijk} \Leftrightarrow \mathsf{InSameL}_{ijk} \wedge \mathsf{Past}\left(\mathsf{Sep}_{ij} > close, 1\right)$
9.$\mathsf{Leave}_{ijk} \Leftrightarrow \mathsf{Past}(\mathsf{InSameL}_{ijk}, 1) \wedge p_i \neq L_k \wedge p_j \neq L_k$
10.$\mathsf{Contact}_{ijk} \Leftrightarrow \mathsf{InSameL}_{ijk} \wedge \mathsf{moving}_{R_j}$
11.$\forall R_j$:
  $a.v_i = \texttt{none} \Leftrightarrow \mathsf{Lasted}(p_j = L_k, 2)$
  $b.v_j = \texttt{low} \Leftrightarrow \mathsf{Lasted}(p_j = L_k, 1) \wedge \mathsf{past}(p_j, 2) \neq L_k$
  $c.v_j = \texttt{mid} \Leftrightarrow \mathsf{past}(p_j, 1) \neq p_j$
  $d.v_j = \texttt{high} \Leftrightarrow \mathsf{past}(p_j, 2) \neq \mathsf{past}(p_j, 1) \neq p_j$
  $e.\mathsf{past}(p_j, 1) = p_j \vee \mathsf{Adj}(\mathsf{past}(p_j, 1), p_j)$
12.$\mathsf{Sep}_{fingers,arm} \in \{\texttt{close}, \texttt{mid}\}$

Action 1. Robot moves to bin from an initial point (all other actions has not started yet).
  Pre-condition 1 $\Leftrightarrow p_{base} =$ initial_point $\wedge \forall x_{\neq 1} : a_x^{sts} = \texttt{ns}$
  Post-condition 1 $\Leftrightarrow p_{base} = p_{bin}$

Action 2. Robot arm is stretching out towards bin right after action 1 is done.
  Pre-condition 2 $\Leftrightarrow a_1^{sts} = \texttt{dn} \wedge \neg\mathsf{moving}_{base} \wedge \neg\mathsf{moving}_{ee}$
  Post-condition 2 $\Leftrightarrow \neg\mathsf{moving}_{base} \wedge p_{ee} =$ bin

and velocity, that are very important in evaluation of the danger of a contact, are

[3] The full formal model is available on github/safer-hrc

captured by the model via two variables whose variations are descretized with four possible values `none`, `low`, `mid`, `high`. Temporal modeling allows for creating a meaningful and smooth fluctuation between these values. For example, the value of velocity cannot jump from `none` to `high`. These variations are modeled in formulae 11 of Table 2. The same formulae hold also for the force.

Another important role of time in modeling, is for reproducing the executing actions of human and robot that together create the full executing task (i.e., job). As we have described in Section 2.3, a more thorough analysis would be provided by defining a rough sense of sequencing between different actions of a task, but not enforcing a static workflow. This is what we do by describing each action with a pair of pre and post conditions which are combinations of temporal and spatial requirements such as required positions of objects, or actions that should have been terminated beforehand. They give a realistic realization to the actions (e.g., human cannot pick a workpiece if he or she is not in front of bin, thus pre-condition of bin-picking action is the correct position for the operator) but generate and explore dynamically all the meaningful and possible workflows (e.g., what happens if operator instead of continuing the bin-picking, stops and switches to inspection action, ...).

The model also contains definitions of physical hazards according to [16] and consistent risk values [17] for detecting the harmful contacts. Here we do not provide detail on the modeling of hazards and risks for the sake of brevity. It suffices to say that each hazard, based on the severity of the harm it could cause and its occurrence frequency, is assigned a risk value among $\{0, 1, 2\}$. If the value of risk is 0, everything is good and risk value equal to 1 is negligible. Otherwise, if there is a risk with value equal to 2, the human operator is in danger. Hence, the latter situation needs to be avoided.

Analysis Scenarios and verification

Scenario A: Detecting Hazards. The main requirement of this case-study was to verify the physical safety of the human operator while working with the robot system. The physical safety itself needs to be interpreted in a general and standard way, and that is why the well-known industrial standards such as [16,15,17] are used in modeling the system. The formal verification procedure is supposed to detect and highlight any possible situation in which the movements of human and robot would violate the constraints implicated by the standards, meaning detecting cases in which risk value is 2. The formula below states the above property, which states for each hazardous situation involving human part i, robot part j, in layout k, the risk should be below 2.

$$\forall BP_i.\forall R_j.\forall L_k : \mathsf{Alw}(\mathsf{risk}_{ijk} \leqslant 1) \tag{7}$$

The verification has been applied via a model checker called Zot [36] that reports the state of the system at each instant of time (e.g., positioning of the human and robot, the state of task execution, the criticality of human and robot interaction,...).

Scenario B: Detecting Human Errors at Runtime Physical Safety of human is a critical issue that needs to be considered in early stages of design. Anticipating hazardous situations and a proper remedy for them are usually tackled during design. However, not every situation is predictable in advance. For example, as discussed in Section 2.3, human operator and her presence brings uncertainty in to the picture and one could never fully predict her behavior during the execution. In order to make the model closer to a real human, we have generated a erroneous behavior model [1] which includes formalized definition of the most frequent human errors that could influence the workflow, which are temporal and spatial oriented. It allows for a model that is applicable also for runtime verification.

4 Conclusions

In this paper, we discussed why formal modeling and verification are needed in designing and operating CPSs, to offer assurances about their dependable use in many practical application areas in which they are increasingly deployed. We also discussed why and how modeling and verification methods need to accommodate the specific new requirements arising from interaction of computing elements with the physical environment, which is typical of CPSs. In particular, we focused on CPSs where the notion of *space* in which the system operates and *time* constraining operations are key. We also stressed how uncertainty, at different levels, heavily affects CPSs and asks for new approaches to formal methods that break the traditional boundary between design time and runtime. We also presented how in our past work we provided solutions to these problems, and case studies we developed in which we such solutions were applied.

The work we presented here, however, can only be viewed as a solution to some of the problems we need to face when formal methods are applied in the design and operation of CPSs. A community effort is needed to consolidate methods and provide both support tools and libraries including application-independent components and open to extensions to ad-hoc components specialized towards single application fields.

References

1. Askarpour, M., Mandrioli, D., Rossi, M., Vicentini, F.: Formal model of human erroneous behavior for safety analysis in collaborative robotics. Robotics and Computer-Integrated Manufacturing **57**, 465 – 476 (2019)
2. Baheti, R., Gill, H.: Cyber-physical systems. The impact of control technology **12**(1), 161–166 (2011)
3. Bures, T., Weyns, D., Schmerl, B.R., Tovar, E., Boden, E., Gabor, T., Gerostathopoulos, I., Gupta, P., Kang, E., Knauss, A., Patel, P., Rashid, A., Ruchkin, I., Sukkerd, R., Tsigkanos, C.: Software engineering for smart cyber-physical systems: Challenges and promising solutions. ACM SIGSOFT Software Engineering Notes **42**(2), 19–24 (2017)

4. Cheng, B., Lemos, R.D., Giese, H., et al.: Software engineering for self-adaptive systems: A research roadmap. In: Software engineering for self-adaptive systems, pp. 1–26. Springer (2009)
5. Ciancia, V., Grilletti, G., Latella, D., Loreti, M., Massink, M.: An experimental spatio-temporal model checker. In: International Conference on Software Engineering and Formal Methods. pp. 297–311. Springer (2015)
6. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: IFIP International Conference on Theoretical Computer Science. pp. 222–235. Springer (2014)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT press (1999)
8. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation-part i: Basic concepts and double pushout approach. In: Handbook of Graph Grammars. pp. 163–246 (1997)
9. Eaton, C.M., Chong, E.K., Maciejewski, A.A.: Multiple-scenario unmanned aerial system control: A systems engineering approach and review of existing control methods. Aerospace **3**(1), 1 (2016)
10. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Berlin, H. (ed.) . Springer Berlin Heidelberg, pp. 214–238 (2013)
11. Furia, C.A., Mandrioli, D., Morzenti, A., Rossi, M.: Modeling Time in Computing. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2012)
12. Galton, A.: A generalized topological view of motion in discrete space. Theoretical Computer Science **305**(1), 111–134 (2003)
13. Godfrey, M.W., German, D.M.: The past, present, and future of software evolution. In: Frontiers of Software Maintenance, 2008. FoSM 2008. pp. 129–138. IEEE (2008)
14. Gröger, G., Kolbe, T.H., Czerwinski, A., Nagel, C., et al.: Opengis city geography markup language (citygml) encoding standard, version 1.0. 0 (2008)
15. ISO 10218-1: Robots and robotic devices – Safety requirements for industrial robots – Part 1: Robots. International Organization for Standardization, Geneva, Switzerland (2011)
16. ISO 10218-2: Robots and robotic devices – Safety requirements for industrial robots – Part 2: Robot systems and integration. International Organization for Standardization, Geneva, Switzerland (2011)
17. ISO 12100: Safety of machinery – General principles for design – Risk assessment and risk reduction. International Organization for Standardization, Geneva, Switzerland (2010)
18. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
19. Lee, E.A.: Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). pp. 363–369 (May 2008)
20. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press (2009)
21. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In: 14, , New York, NY, USA ACM. pp. 3–14. Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ICPE (2014)
22. Rajhans, A., Cheng, S.W., Schmerl, B., Garlan, D., Krogh, B.H., Agbi, C., Bhave, A.: An architectural approach to the design and analysis of cyber-physical systems. Electronic Communications of the EASST **21** (2009)

23. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: Proc. of the 47th Design Automation Conference. pp. 731–736. ACM (2010)
24. Sánchez-García, J., García-Campos, J.M., Toral, S., Reina, D., Barrero, F.: An intelligent strategy for tactical movements of UAVs in disaster scenarios. International Journal of Distributed Sensor Networks **2016** (2016)
25. Tan, J.T.C., Duan, F., Zhang, Y., Watanabe, K., Kato, R., Arai, T.: Human-robot collaboration in cellular manufacturing: Design and development. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 29–34
26. Tsigkanos, C., Kehrer, T., Ghezzi, C., Pasquale, L., Nuseibeh, B.: Adding static and dynamic semantics to building information models. In: Proc. of the 2nd Intl. Workshop on Software Engineering for Smart Cyber-Physical Systems. pp. 1–7. ACM (2016)
27. Tsigkanos, C., Pasquale, L., Ghezzi, C., Nuseibeh, B.: On the interplay between cyber and physical spaces for adaptive security. IEEE Transactions on Dependable and Secure Computing **PP**(99), 1–1 (2016)
28. Tsigkanos, C., Kehrer, T., Ghezzi, C.: Architecting dynamic cyber-physical spaces. Computing **98**(10), 1011–1040 (2016)
29. Tsigkanos, C., Kehrer, T., Ghezzi, C.: Modeling and verification of evolving cyber-physical spaces. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, 2017. pp. 38–48 (2017)
30. Tsigkanos, C., Nenzi, L., Loreti, M., Garriga, M., Dustdar, S., Ghezzi, C.: Inferring analyzable models from trajectories of spatially-distributed internet-of-things. In: IEEE/ACM Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, Canada, May 25-26, 2019. IEEE Computer Society (2019)
31. Tsigkanos, C., Pasquale, L., Ghezzi, C., Nuseibeh, B.: On the interplay between cyber and physical spaces for adaptive security. IEEE Trans. Dependable Sec. Comput. **15**(3), 466–480 (2018)
32. Vogel-Heuser, B., Feldmann, S., Folmer, J., Kowal, M., Schaefer, I., Ladiges, J., Fay, A., Haubeck, C., Lamersdorf, W., Lity, S., et al.: Selected challenges of software evolution for automated production systems. In: Industrial Informatics, 2015 IEEE 13th Intl. Conf. pp. 314–321. IEEE (2015)
33. Wang, S., Wan, J., Li, D., Zhang, C.: Implementing smart factory of industrie 4.0: an outlook. International Journal of Distributed Sensor Networks **12**(1) (2016)
34. Xie, J., AI-Emrani, F., Gu, Y., Wan, Y., Fu, S.: UAV-carried long distance wi-fi communication infrastructure. In: AIAA Infotech@ Aerospace (2016)
35. Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., Aberer, K.: Semantic trajectories: Mobility data computation and annotation. ACM Transactions on Intelligent Systems and Technology (TIST) **4**(3), 49 (2013)
36. Zot: a bounded satisfiability checker. available from github/fm-polimi/zot (2012)