A Methodology for the Design and Deployment of Distributed Cyber-Physical Systems for Smart Environments

Giacomo Tanganelli[a,*], Luca Cassano[b], Antonio Miele[b], Carlo Vallati[a]

[a] *Università di Pisa – Lungarno Pacinotti 43, 56126 Pisa, Italy*
[b] *Politecnico di Milano – P.zza Leonardo Da Vinci 32, 20133 Milano, Italy*

Abstract

The pervasiveness and the growing processing capabilities of mobile and embedded computing systems are leading to a shift from the Internet of Things (IoT) paradigm to the Fog computing scenario where the environment is instrumented with high-performance computing in the proximity to cyber-physical systems. The design of such systems requires an accurate planning, on the one hand, to ensure that specific application requirements will be properly met at run-time, and, on the other hand, to minimize the system's monetary costs. In this paper we present a methodology for an automated design and deployment of distributed cyber-physical systems into smart environments. We propose an engine based on a Mixed Integer Linear Programming (MILP) formulation which takes in input a planimetry of the environment and a description of the applications and, based on a repository of available processing boards, identifies the cost-optimized instantiation of the processing architecture and the corresponding distribution of the application functionalities. By comparing our proposal with the existing methodologies that address similar problems we can highlight the following novelties: i) we address a system architecture composed of heterogeneous devices, ii) we adopt a realistic model of the environment, and iii) we perform a joint co-exploration of architecture instantiation and applications mapping. An experimental evaluation, considering a smart office case study, demonstrates the potential of the proposed approach in minimizing the overall system monetary cost around 42% w.r.t. a baseline approach not exploiting planimetry information. Such results have been also confirmed by an extensive experimental campaign using synthetic problems, which also highlighted how the execution times of the optimization process are affordable for the design-time process.

Keywords: Distributed Cyber-Physical Systems; Smart Environments; Mixed Integer Linear Programming.

1. Introduction

Fog computing [7, 8] is commonly considered as a crucial enabling technology for future Internet of Things (IoT) systems. Fog computing extends the original Cloud computing architecture, in which processing and storage capabilities are confined into datacenters, with an additional layer, i.e. the Fog layer, installed in the proximity to cyber-physical systems. Such Fog layer enables the execution of the application logic on processing nodes in direct connection with IoT devices, thus supporting information process and analysis in a short distance from where the data is produced, see Figure 1.

Low communication latency between applications and IoT devices is crucial to support applications that have stringent delay requirements [17] and, therefore, not supported in common Cloud computing systems in which data is offloaded to datacenters throughout the Internet. As discussed in [30], closed-loop automation logic [34], augmented reality services [2] or real-time video analytics [31], hospital emergency systems [3], home automation [13] are only a few examples of systems for smart environments that can be enabled through Fog computing.

The installation of Fog nodes, which rely on high performance embedded computing boards, to support cyber-physical systems in smart environments, however, requires an accurate planning. Although, small environments, e.g. a small office or a house, might be

*Corresponding author
Email addresses: giacomo.tanganelli@unipi.it (Giacomo Tanganelli), luca.cassano@polimi.it (Luca Cassano), antonio.miele@polimi.it (Antonio Miele), carlo.vallati@unipi.it (Carlo Vallati)

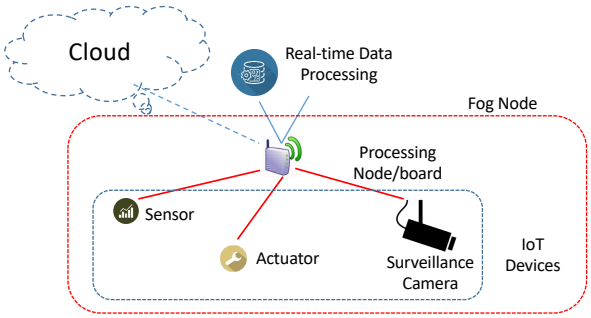


Figure 1: Fog computing scenario.

served satisfactorily by only upgrading the few existing computing nodes, such as existing network equipment like gateways or routers, larger and more complex environments, such as an office building comprising many floors or an university campus covering a large district, will require the installation of a large number of new Fog nodes. This activity has to consider the specific application requirements of the cyber-physical system, both in terms of communication capabilities (i.e. minimum bandwidth) and the coverage of specific geographical locations to interact with. These requirements are considered essential in many of the Fog computing use-cases envisioned by the OpenFog consortium [20] and by several companies [1, 12, 21]. For instance, in the *smart building* use-case a smart alert system [23] requires the complete coverage of the area in order to ensure that data from all the sensors are collected and analyzed. On the other hand, a *smart video surveillance* system requires that processing nodes are placed not only to meet geographical and communication requirements but also considering processing ones to ensure that the flow of images from cameras is captured and elaborated with the required throughput.

Unfortunately, a systematic analysis of the literature on Fog computing allowed us to conclude that past approaches either consider different issues, such as the definition of resource management middlewares and the runtime workload balancing and distribution [4, 25, 33], or when considering the system design, they consider a sub-part of the problem. In particular, design approaches for Fog computing systems generally consider the workload mapping on a given architecture [9, 26, 27] or, when instantiating the architecture, rarely consider the geographical position of the Fog nodes in the optimization [10, 11].

Given these motivations, we present in this work a methodology to automate the design and deployment of cyber-physical systems for smart environments. The goal of the proposed approach is to automatically identify a suitable architecture of the cyber-physical system and the corresponding applications' workload distribution in order to minimize the architectural cost and ensure that requirements of each application, such as processing resource requirements and cyber-physical interaction ones, are met. Specifically, we aim at defining an approach for the placement of the Fog processing nodes based on the planimetry of an environment with candidate positions and the requirements of the smart application. The goal of the proposed approach is to minimize the overall deployment cost by selecting the optimal placement and configuration of the Fog nodes. To this aim, we propose an engine based on a Mixed Integer Linear Programming (MILP) formulation which takes in input: 1) a planimetry of the environment with the set of candidate positions for the placement of processing nodes, 2) a description of the workload representing the smart applications, and 3) a repository of available processing boards, i.e. embedded systems available for deployment as Fog processing nodes. The engine produces a cost-optimized instantiation of different processing boards in a sub-set of available positions in the planimetry to compose a distributed architecture and a corresponding workload distribution. The architecture will be capable of providing the necessary computation and communication resources to the executed workload and, at the same time, mapping each application in the geographical location required for its cyber-physical interaction with the surrounding environment.

To summarize, the contributions of this paper are the following:

- a novel model of the processing node placement problem that includes realistic reference architecture and environmental models, the last one characterized by overlapping geographical locations,
- a MILP formulation that ensures a cost-optimized deployment of Fog nodes to minimize the overall cost while still ensuring that workload requirements are met,
- a performance evaluation of the proposed approach, which also considers a realistic use-case.

Moreover, the proposed approach differs from the past work on the design and deployment of Fog computing system (e.g. [9, 26, 27, 11]) by considering the following aspects:

- the adoption of a reference architecture, built by instantiating various types of heterogeneous devices presenting different performance/cost tradeoffs,

- the joint co-exploration of architecture instantiation and applications mapping, and
- the adoption of a realistic environmental model presenting overlapping geographical locations for cyber-physical interactions of different applications, which provide an additional degree of freedom to opportunistically place processing devices during the joint optimization.

The rest of the paper is organized as follows. Section 2 gives an overview of the related work on the considered problem. Then, Section 3 introduces a detailed presentation of the problem, later modeled in a formal way in Section 4. The design automation engine based on a MILP formulation is proposed in Section 5. The effectiveness of the approach is analyzed in Section 6 by means of a case study, discussed in details, and an extensive evaluation based on synthetic problems. Finally, Section 7 draws the conclusions and presents future work.

2. Related Work

The majority of the literature on Fog computing investigated the optimization of resource provisioning and workload mapping on an already-given architecture; this activity is performed either at runtime or at design time. Some papers [4, 25] propose middlewares for runtime resource management of distributed systems composing the Fog infrastructure and the dynamic mapping of the incoming workload. In contrast, the work here proposed does not deal with runtime resource management; instead, it focuses on system-level design optimization, that is performed at the time of the installation of the cyber-physical system to minimize its monetary cost.

When considering the design-time optimization, in [33], for example, an optimal placement methodology for virtual machines in Fog computing systems is proposed, taking into account the opportunity to deploy multiple replicas to ensure high availability and short response time. In [32], instead, the authors propose a task scheduling algorithm for delay-sensitive applications. Having an industrial use-case in mind, a scheduling model for containerized tasks is designed to assign tasks to nodes ensuring that each one is completed on time and the number of concurrent tasks is optimized. Authors of [19] propose a resource allocation strategy for Fog computing based on priced timed Petri nets aimed at dynamically allocating Fog resources at runtime. In [9] the authors propose a simulation framework for Fog computing able to perform an automated exploration of a workload on an architecture received as input. An exhaustive exploration is performed to identify a solution satisfying applications' computation and communication requirements within the architecture capabilities. Other SystemC simulators for networked embedded systems [24, 6] can be adopted in the considered scenario as they consider all the necessary computation and communication aspects, however, the geographical distribution of nodes is not modeled. A more advanced exploration engine is proposed in [26], where a genetic algorithm has been designed to maximize the number of services that can be deployed in a Fog architecture, which has multiple hierarchical layers to reduce the amount of workload offloaded in the Cloud. Two similar approaches optimizing workload distribution in a Fog-Cloud architecture have been defined to minimize the traffic [27], and also energy consumption and application latency [28]. Finally, an already-instantiated architecture is considered in [18] to optimize the data placement among the various nodes. In conclusion, the main limitations that characterize existing approaches are that the architecture is received as input and it is fixed, i.e. no geographical/planimetric information are included in the architectural/application models.

Architecture instantiation and geographical placement of the processing nodes have been considered in previous works focusing on different contexts, i.e. Wireless Sensor Network (WSN) and IoT scenarios. It is worth mentioning the framework proposed in [22] that performs an automated synthesis of the network and placement of wireless sensors and base stations. The framework considers also reliability issues by introducing architectural redundancies aimed at improving the system's lifetime. Indeed in WSN and IoT scenarios, application mapping is not considered in the design problem; in fact, each application functionality is already bound with a specific IoT node. Therefore the design problem consists only of the architecture instantiation. Then, in [29] a planimetry where IoT nodes are already installed is considered to optimize the placement of services in order to improve the efficiency of communications of a ZigBee network. Another mapping approach considering the topology is proposed in [5], however, the work considers only a simple planimetry composed by disjointed rectangular regions. Finally, a very simple square homogeneous region where nodes have to be placed is considered in [15].

The most relevant recent work for our investigation has been presented in [10, 11], where approaches for the modeling and the synthesis of distributed embedded systems in the Fog computing scenario are proposed. In particular in [11] the authors propose a topology-

Table 1: Qualitative comparison of the proposed approach w.r.t. the past work.

		Approach															
		[33]	[32]	[19]	[9]	[24]	[6]	[26]	[27]	[28]	[18]	[22]	[29]	[5]	[15]	[11]	**PA**
Knob	**Arch. Inst.**								√			√				√	√
	Mapping	√	√	√	√	√	√	√	√	√	√	√	√	√		√	√
Model	**Planimetry**											B	B	B	B	B	ADV.
	Transmission	√			√	√	√	√	√	√	√	√	√	√	√	√	√
Metric	**Workload Perf.**	√	√	√	√	√	√	√	√	√	√	√	√	√		√	√
	Traffic	√			√	√	√	√	√	√	√	√	√	√	√	√	√
	Economic Cost									√			√			√	√
	Power					√			√			√				√	√

aware architecture model and a formal framework to efficiently synthesize the system in terms of architecture instantiation and applications' mapping. The approach mainly focuses on concurrently minimizing the economic cost, the energy consumption and the delay and error rate in the transmissions. However, also this work is mainly focused on the networking aspects while the planimetric model of the environment is not exploited to share Fog nodes among different applications requiring cyber-physical interactions in geographical locations partially or fully overlapping. Instead, our approach also aims at exploiting the possibility to share geographical resources to further minimize the monetary cost of the solution without considering in depth the networking aspects.

Table 1 summarizes the characteristics of the discussed past approaches and compares them against the proposed work. For each paper we highlight the considered action knobs (architecture instantiation and task mapping), the models for planimetry (B: basic or ADV.; advanced) and transmission, and the metrics adopted as optimization goal/constraint (workload performance, traffic, economic cost and power consumption). In conclusion, the proposed approach advances the past works by considering all previous features and introducing an advanced planimetry model that captures cyber-physical interactions with the environment.

3. Problem Description

The problem we consider deals with the design of the computing architecture of a distributed cyber-physical system in a given environment and the subsequent smart workload deployment. As a running example, Figure 2 shows the simple planimetry of a floor of an office building requiring two applications: advanced video surveillance for intrusion and fire detection, and smart music broadcasting services.

In this scenario, we identify two different actors: 1) the architect who designed and supervised the construction of the considered environment, and 2) the engineer of the cyber-physical system who designs the smart computing infrastructure. We assume that the architect has already instrumented and cabled the considered environment to support the installation of processing nodes, which will compose the distributed cyber-physical system, and to provide access points to enable the communication. Such installation points, referred to as *spots*, are depicted with white circles in the planimetry in Figure 2(a). The engineer can therefore select a set of boards from his/her repository of available processing components to be installed in the desired spots in order to build the distributed architecture for the cyber-physical system.

The applications deployed in smart environments generally have a cyber-physical nature, as they require to interact with the surrounding environment. Indeed, a complex application may be composed of several cooperating tasks. Some of the tasks have to sense and/or actuate in different, possibly distant, geographical *zones*; therefore, these tasks need to be executed in the specific zone they need to interact with. Other tasks, such as data aggregation or post-processing, do not present such location-specific requirements and therefore can be executed in any geographical position. In the running example in Figure 2(b), applications are represented as task graphs, and the zones are depicted as dashed rectangles. Each zone is also annotated with the specific application's tasks to be executed in. It is worth noting that different zones may possibly completely or partially overlap. In the example, we would like to perform both video/fire surveillance and music broadcasting in the two rooms, thus leading to two completely overlapping zones (indeed the figure shows a single dashed rectangle). Moreover, the L-shape of the corridor will require the definition of two different partially-overlapped zones for video surveillance, therefore requiring two distinct tasks.

In order to design the system, the engineer has to identify an instantiation of the computing architecture

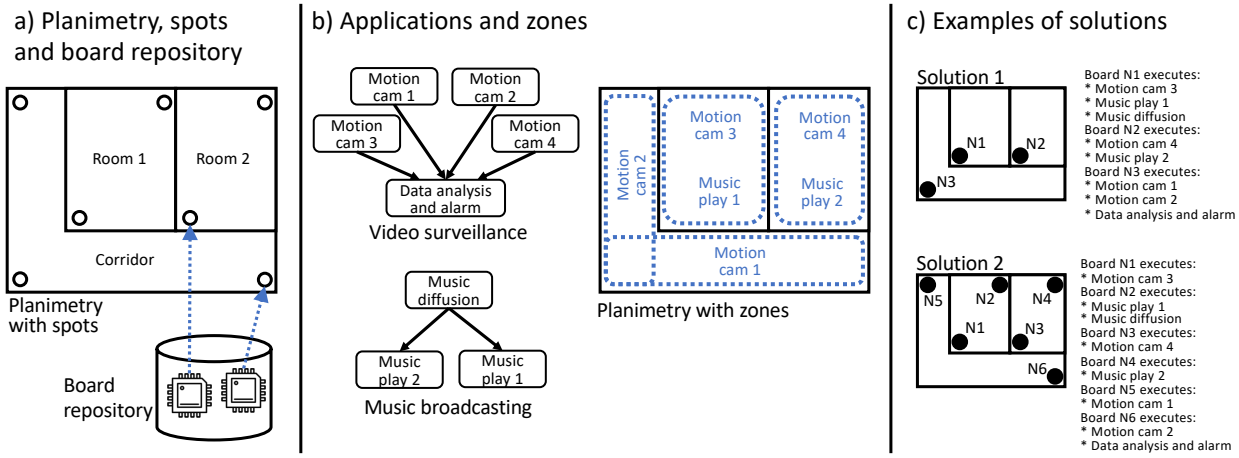


Figure 2: A running example: a floor of a smart office building.

on the available spots and a subsequent deployment of the applications on the installed processing boards. In this scenario, the availability of various types of boards, the presence of several spots where to install the boards and geographical complete/partial overlapping among zones and spots open to a large set of possible alternative solutions to the considered design problem. For the considered example, Figure 2(c) shows two alternative solutions; installed boards are depicted with black circles and the list of tasks executed by each board is reported on the right side. For the corridor different decisions have been adopted in the two solutions; in Solution 1 a single board has been installed in the central angle comprised in both zones (in the bottom-left part of the planimetry) to execute all the tasks, while in Solution 2 two cheaper boards, located in one of the zones each in the opposite corners, have been used to split the workload. A similar analysis may be carried out for each of the two rooms; we may instantiate a single powerful board for hosting all the desired functionalities or partition fire/video surveillance and music broadcasting on two separate cheaper boards. Indeed, the two presented solutions represent the opposite bounds of a design space and between the two ones, various other combinations may be identified to partially exploit zone overlapping and resource sharing. The decision on how to select the solution is therefore driven by some optimization function, that may be for instance the monetary cost of the system capable at providing the desired functionalities. In the example, the designer will select one of the two solutions (or some other intermediate one) based on the higher convenience in instantiating a lower number of powerful boards or a larger number of cheaper boards. As a conclusion, these peculiarities of the cyber-physical nature of the system open new possibilities for optimizing the system during the synthesis process. Moreover the larger the size of the considered problem (in terms of dimension of the board repository, number of applications' tasks, number of zones and number of spots), the larger the dimension of the depicted design space, that may assume a considerably huge size.

4. System Model

Based on the design problem discussed in the previous section, we have defined the following system model representing the environment planimetry, the architecture to be instantiated and the workload to be deployed. Then, the system design problem is formulated presenting the synthesis constraints and the optimization function.

Environment planimetry. The standard environment planimetry provided by the architect has been annotated with two additional aspects, called spots and zones, respectively. A *spot*, representing a possible position where a processing board can be installed, is formally defined with the tuple:

$$s_i =< (x, y), b_{up}, b_{down} > \quad (1)$$

representing the coordinates (x, y) of the spot in the planimetry, and information regarding the available connection, in particular the provided up/downlink bands. If the planimetry consists of various floors, coordinates will include an additional field. Moreover, also 3D models can be considered for the planimetry; a z coordinate will be accordingly added. Finally, for the sake

of simplicity, a single connection type is considered for each spot; however, the model can be extended with a vector to consider multiple, selectable connections.

A *zone* of cyber-physical interaction of a task is modeled as a polygon in the planimetry. Formally, the zone is represented with the tuple:

$$z_i =< coords, spots > \tag{2}$$

where $coords$ is a non-empty list of coordinates (x, y) forming the polygon, and $spots$ a non-empty list of positions s_i in which the involved activity can be executed on the zone itself.

Application. As common practice in the considered scenario (e.g., [11, 27]), an *application* (or a set of applications) is modeled as directed acyclic graph $G_{appl} = (T, D)$, where T is the set of *tasks* t_i composing the distributed application and D is the set of edges $d_{i,j}$, representing data dependency among tasks t_i and t_j.

Each task executes a functionality on a specific zone by interacting with the environment by means of a set of sensors and actuators. It derives that the *task* t_i is modeled in terms of the tuple:

$$t_i =< type, z, sens, acts > \tag{3}$$

where $type$ represents the functionality, z the zone where to execute the task, and $sens$ and $acts$ the lists of required sensors and actuators. The performance of the task is tightly connected to each board that will be in charge of its execution; therefore, the performance model is discussed later in the architecture's model.

The *edge* $d_{i,j}$ is only characterized with the amount of exchanged data per unit of time between two tasks, i.e. the required band:

$$d_{i,j} =< b_{req} > \tag{4}$$

Architecture. In this scenario, we assume to have a repository of boards that can be installed in the available spots to realize the distributed system architecture running the required workload. Each board b_i represents a class of Fog nodes and is characterized in terms of its processing resources (in particular, the number of cores in the CPU, the size of RAM and the one of the disk), the lists of peripherals for sensing and actuating on the surrounding environment, and its monetary cost:

$$b_i =< cpu, ram, disk, sens, acts, cost > \tag{5}$$

Moreover, since we expect to have a heterogeneous set of boards, the execution footprint of each task t_i on each board b_j is annotated in a performance table, i.e. a map storing storing for each pair (t_i, b_j) the required resources:

$$perf(t_i, b_j) :< t_i, b_j >\rightarrow< cpu, ram, disk > \tag{6}$$

it is worth noting that a task may also require just a fraction of a CPU core based on the concept of CPU quota provided by the operating system scheduler to each process when running a multiprogrammed workload. Moreover, if a task cannot be executed on a specific board, $perf$ will contain a $< +\infty, +\infty, +\infty >$ tuple.

Synthesis problem formulation. Given the above discussion, a system design is defined by instantiating a non-empty set of nodes N, each one n_i consisting in 1) the installation of a specific board in the selected spot, and 2) in the mapping of all the tasks of the application, each one on a single node of the instantiated architecture. Therefore, the node is formally defined as:

$$n_i =< b, s, tasks > \tag{7}$$

being $tasks$ the non-empty set of tasks mapped on the current node. Do note that the board here represents a class of Fog nodes; therefore, we may install different instances of the same board in different nodes of the same architecture.

In the definition of a solution in terms of a set of nodes N, a set of constraints has to be guaranteed:

- At most a node can be instantiated in each spot[1]:

$$\forall n_i \in N,\ \forall n_j \in N \text{ with } i \neq j : n_i.s \neq n_j.s \tag{8}$$

- Each task t_j has to be mapped on a single node n_i:

$$\forall t_j \in T\ \exists n_i \in N : t_j \in n_i.tasks \tag{9}$$

$$\forall n_i \in N,\ \forall n_j \in N \text{ with } i \neq j : n_i.tasks \cap n_j.tasks = \emptyset \tag{10}$$

- Each task t_j has to be mapped in a node n_i installed in a spot included in the required zone $t_j.zone$:

$$\forall n_i \in N,\ \forall t_j \in n_i.tasks : n_i.s \in t_j.z.spots \tag{11}$$

- Each node n_i has to provide all the sensors and actuators required by each task mapped on it, i.e.

[1] Do note that here we use the dot notation to refer to a specific attribute of a tuple.

$t_j \in n_i.tasks$:

$$\forall n_i \in N,\ \forall t_j \in n_i.tasks : \\ n_i.sens \subset t_j.sens \ \wedge \ n_i.acts \subset t_j.acts \quad (12)$$

- Each node n_i has to provide the processing resources at least equal to the sum of requests of all the mapped tasks $n_i.tasks$:

$$\forall n_i \in N : \left(n_i.cpu \geq \sum_{t_j \in n_i.tasks} t_j.cpu \right) \quad (13)$$

  For the sake of brevity only CPU resource is reported for the formula; the same constraint applies to RAM and disk resources.

- The node n_i has to provide the communication resources at least equal to the sum of requests of all the mapped tasks $n_i.tasks$. Therefore, assuming $\mathcal{D}_{out\_j}$ be the set of edges $d_{j,k}$ outgoing task t_j, the constraint can be formulated as:

$$\forall n_i \in N : \left(n_i.s.b_{up} \geq \sum_{d_{j,k} \in D_{out\_j}} d_{j,k}.b_{req} \right) \quad (14)$$

  The above formula refers to the uplink band. The formulation requires a similar constraint for the downlink band.

The goal of the defined design problem is to identify an implementation consisting of a set of nodes N minimizing the architecture cost; the overall cost is computed as the sum of the costs of all instantiated boards in the various nodes:

$$cost_{arch} = \sum_{n_i \in N} n_i.b.cost \quad (15)$$

As a final note, the model presented in this paper focuses on the design of the distributed cyber-physical system on which the Fog layer relies. Although left as future work, it is important to highlight that the proposed model can be easily extended to support applications whose implementation is distributed between Fog and Cloud. To this aim, additional formulas can be introduced to model the interaction between the Fog and Cloud layers. On one side, one or more additional constraints are needed to take into account the application requirements in the communication between the Fog and the Cloud layers, i.e. the minimum uplink bandwidth required to offload the data to the Cloud towards application components that perform historic data collection or big data analysis. On the other, the cost function needs to be modified in order to include the costs of deploying some parts of the application in the Cloud.

Table 2: MILP variables, sets and parameters

Sets	
S	set of available spots ($\|S\| = s$)
T	set of task ($\|T\| = t$). Each element t_j represents a specific task mapped in a specific spot s_k
B	set of board instances ($\|B\| = b$). Each element b_i represents a specific board installed in a specific spot s_k
Parameters	
$Bcpu_i$	number of cores of board i
$Bram_i$	ram space of board i
$Bdisk_i$	disk space of board i
$Bsen_i$	list of sensors of board i
$Bact_i$	list of actuators of board i
$Bcost_i$	cost of board i
Sup_k	uplink bandwidth of spot k
$Sdown_k$	downlink bandwidth of spot k
up_j	uplink bandwidth requirement of task j
$down_j$	downlink bandwidth requirement of task j
$cpu_{i,j}$	number of cores required by task j on board i
$ram_{i,j}$	ram space required by task j on board i
$disk_{i,j}$	disk space required by task j on board i
Pre-Processing Parameters	
$m_{i,j}$	binary parameter set to 1 if and only if the task j can be deployed on board i
$z_{j,k}$	binary parameter set to 1 if and only if the task j can be deployed in spot i
$cpu_{i,k,j}$	number of cores required by task j on board i in spot k
$ram_{i,k,j}$	ram space required by task j on board i in spot k
$disk_{i,k,j}$	disk space required by task j on board i in spot k
Variables	
$x_{i,k,j}$	binary variable set to 1 if and only if task t_j is hosted on board b_i installed in the spot s_k
$y_{i,k}$	binary variable set to 1 if and only if the board b_i is installed in the spot s_k

5. Design Space Exploration Engine

The system model specified in the previous section is a Generalized Assignment Problem (GAP) [16], that is generally solved by means of a MILP formulation. The variables, sets and parameters of the formulation are listed in Table 2, whereas the model constraints and optimization function are summarized in Table 3.

Formula C1 in Table 3 defines the objective function, that is to minimize the overall system's cost, as defined in Equation 15. To define the constraint, we introduce a decision variable $y_{i,k}$ to state if a board i is installed in the spot k.

Table 3: MILP model constraints and optimization function

Cost function	
C1	$\min \sum_{i \in B} \sum_{k \in S} Bcost_i \cdot y_{i,k}$
Task allocation	
C2	$\sum_{i \in B, k \in S} x_{i,k,j} = 1 \; \forall j \in T$
Board placement	
C3	$\sum_{i \in B} y_{i,k} \leq 1 \; \forall k \in S$
C4	$y_{i,k} \leq x_{i,k,j} \; \forall i \in B, \forall k \in S, \forall j \in T$
Cyber-physical requirements	
C5	$(m_{i,j} + z_{j,k}) \cdot x_{i,k,j} = 0 \; \forall i \in B, \forall k \in S, \forall j \in T$
Computation requirements	
C6	$\sum_{j \in T} cpu_{i,k,j} \cdot x_{i,k,j} \leq Bcpu_i \; \forall i \in B, \forall k \in S$
C7	$\sum_{j \in T} ram_{i,k,j} \cdot x_{i,k,j} \leq Bram_i \; \forall i \in B, \forall k \in S$
C8	$\sum_{j \in T} disk_{i,k,j} \cdot x_{i,k,j} \leq Bdisk_i \; \forall i \in B, \forall k \in S$
Communication requirements	
C9	$\sum_{j \in T} up_j \cdot x_{i,k,j} \leq Sup_k \; \forall i \in B, \forall k \in S$
C10	$\sum_{j \in T} down_j \cdot x_{i,k,j} \leq Sdown_k \; \forall i \in B, \forall k \in S$

The feasibility of the solution is modeled by the rest of the formulas in Table 3. Constraint C2 ensures that all the task instances are allocated to one and only one board, as described by Equations 9 and 10. In such a constraint, the decision variable $x_{i,k,j}$ is used to annotate a task j hosted on a board i installed in the spot k. Then, Constraints C3-C4 are the *Board placement* constraints and are exploited to define the relation between $x_{i,k,j}$ and $y_{i,k}$. Specifically, the former ensures that only one board is allocated in a specific spot (as stated by Equation 8), whereas the latter is used to count the cost of an instantiated board only once, independently from the number of allocated tasks.

Constraint C5 guarantees the cyber-physical requirements, and therefore that each task is mapped in the required zone on a node having necessary sensors and actuators. Actually, Equations 11 and 12 cannot be easily expressed in a MILP formulation since includes set inclusion operations. Therefore, in order to simplify the formulation, two additional context matrices Z having size $t \times s$ and M having size $b \times t$ are introduced. Specifically, each element of Z states if a task t_j cannot be mapped on a given spot s_k since s_k is not included in the required zone:

$$z_{j,k} = \begin{cases} 0 & if\ task\ j\ can\ be\ deployed\ in\ spot\ k \\ 1 & otherwise \end{cases} \quad (16)$$

and, each element of M if a task t_j can be executed on board b_i, since the board does not contain all required sensors and actuators:

$$m_{i,j} = \begin{cases} 0 & if\ task\ j\ can\ be\ hosted\ on\ board\ i \\ 1 & otherwise \end{cases} \quad (17)$$

Matrices Z and M are computed in a pre-processing phase before the MILP optimization by performing an exhaustive exploration and analysis of all the combinations between the list of tasks T, the list of spots S, and the lists of boards B. As a consequence, the sum in the formula will return the number of violations to the cyber-physical requirements and we require it to be equal to zero.

Subsequent Constraints C6-C8 are exploited to verify that each board has enough CPU, RAM and disk space for all the mapped tasks, as stated in Equation 13.

It is worth noting that we compressed the MILP formulation by merging constraints C5 and C6 as follows. Based on the additional matrices the execution footprint of each task t_j on each board b_i deployed in a spot s_k is modeled as follows:

$$cpu_{i,k,j} = \begin{cases} cpu_{i,j} & if\ m_{i,j} = 0 \vee z_{j,k} = 0 \\ +\infty & otherwise \end{cases} \quad (18)$$

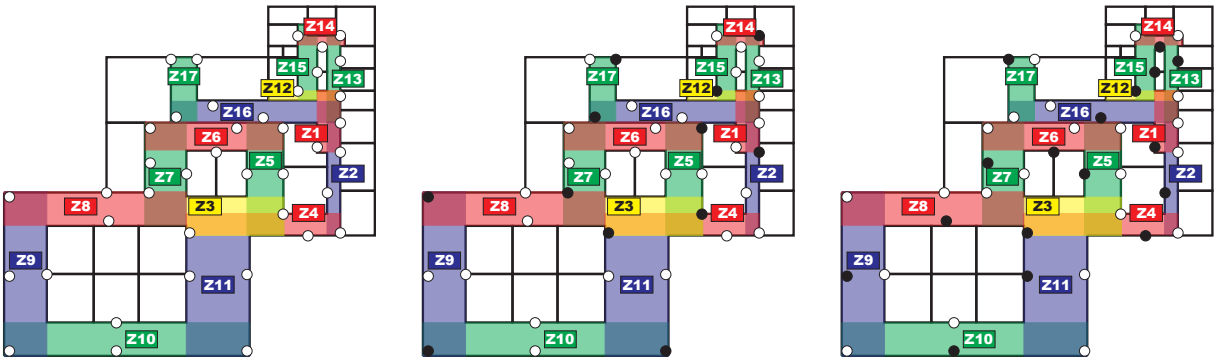
Thus in case a task cannot be allocated to a board or it cannot be deployed in a specific spot, the CPU cost is equal to $+\infty$ and therefore the constraint on the maximum CPU, i.e. C6, will be never fulfilled.

Similar considerations are drawn for the communication requirements; Constraints C9 and C10 specify the available bandwidth of each spot, to check the upwards and downwards available bandwidth, respectively, as stated in Equation 14.

The solution provided by the MILP solver is used to derive the optimum system architecture. Specifically, the obtained solution is the one that minimizes the overall deployment cost according to the equation specified in Formula C1. It is worth noting that, for some spot there may be no board capable of providing all the processing resources and peripherals of the group of mapped tasks, thus leading to an unfeasible solution.

6. Experimental Evaluation

The proposed methodology has been evaluated through an extensive experimental campaign comprising both realistic problems and synthetic ones. In order to analyze the performance in a realistic scenario, we first considered an office building use case. In this case study, two smart services, i.e. fire detection and video



(a) Office's floor planimetry (b) Our solution (c) Basic solution

Figure 3: Planimetry of the smart office case study.

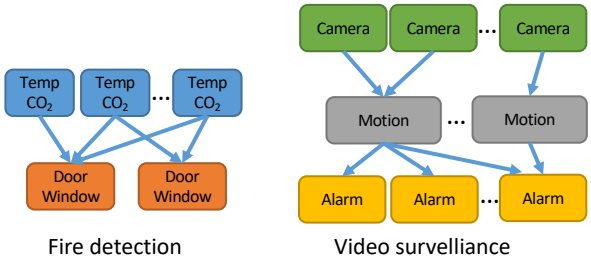


Figure 4: Application task graphs of the smart office case study.

surveillance, are considered for deployment in an office floor planimetry. Then, an extensive experimental campaign has been carried out in order to evaluate the performance on a wide range of different scenarios. In this last experiment, different synthetic problems have been randomly generated.

As discussed in Section 2, this work addresses a different problem w.r.t. the past approaches; as a consequence it is not possible to perform a direct comparison against anyone of them. As highlighted in Table 1, only few works consider both architecture instantiation and task mapping problems. Moreover only one of them considers the economic cost in its metrics [11], but it mainly focuses on the networking aspects. Therefore, we compared our solution against a baseline approach that considers all the zones as separate, i.e. it tackles the problem in an independent manner for each zone.

We implemented a prototype of the proposed automation framework by means of the following technologies: we employed the standard IBM ILOG CPLEX optimization solver [14], for the MILP formulation and we implemented the preprocessing scripts and related wrapper and utility modules for the input/output in Python. We performed two different experimental campaigns; the former considered a case study to show in detail the results of the synthesis process and compare the solution against a baseline implementation. The latter employed a large set of synthetic problems to perform a systematic analysis of the peculiarities of the solutions and design process. All the experiments have been performed on an Intel® Core™ i7-4770 CPU @ 3.40GHz, and 16 GB of RAM running a 64-bit Linux operating system.

6.1. Case Study: a Smart Office Building

We defined a case study where to apply the proposed approach consisting in a floor of an office building requiring two smart services: fire detection and video intrusion surveillance. Figures 3(a) and 4 report the task graphs of the two applications and the planimetry of the building's floor.

Regarding the planimetry, we mainly focused on the corridors and we placed a set of 40 spots (represented as white circles in Figure 3(a)), both in the angles and in the center of each single corridor. Then, we defined 17 zones which various tasks have to interact with (depicted as rectangles in the figure by using different colors for the sake of readability). Each zone includes on average 4 different spots, possibly shared. Finally, we specified a WiFi connection as the communication technology for all the spots. It is worth mentioning that we focused on the corridors since they present an irregular form with several angles thus leading to a more interesting optimization problem for our approach. On the other hand, each room can be considered as a self-standing area being disconnected from the rest of the floor; as a consequence, the related architecture deployment and task mapping can be considered as a separate optimization sub-problem, presenting a trivial solution.

The video surveillance is composed of a set of tasks acquiring pictures from the camera, and two subsequent stages to perform motion detection and to, in case of intrusion, interact with the alarm speakers. The fire detection application instead has a set of tasks to collect data from CO_2 and temperature sensors to be then aggregated and analyzed in order to fire an alarm and to close all doors and windows. The overall task graph consists of 60 different tasks. Since, the two applications require to monitor uniformly the entire floor, tasks requiring sensing/actuation interactions with the environment are uniformly distributed in all the corridors.

We characterized a repository of boards by considering various real-world commercial solutions, namely Raspberry Pi 3, Hardkernel Odroid XU3, Asus Tinker and an Intel Mini-PC NUC. Such solutions represent various options with different prices and specifications, e.g. with RAM ranging from 1024 to 3072 and different CPU options from dual-core to octa-core. In order to consider different peripheral configurations of the boards, we also defined different setups, each one with varying connected sensors and actuators. As result, the repository consists of 15 different boards, whose cost spans between 25USD and 519USD as can be seen in Table 4.

Table 4: Board Repository

Name	CPU	RAM	Disk	Sensors	Actuators	Cost
Raspberry0	4	1024	6400	Camera, CO_2, Temp	Door, Window, Speaker	75
Raspberry1	4	1024	6400	Camera	Door, Window	55
Raspberry2	4	1024	6400	Temp, CO_2	Door, Window, Speaker	45
Raspberry3	4	1024	6400		Door, Window	35
AsusTinker BoardS0	4	2048	12800	Camera		135
AsusTinker BoardS1	4	2048	12800	Camera, CO_2, Temp	Speaker	174
AsusTinker BoardS2	4	2048	12800	Temp, CO_2	Speaker	110
Odroid0	8	2048	3200	Camera, CO_2, Temp		89
Odroid1	8	2048	3200	Camera		59
Odroid2	4	512	800	Temp, CO_2	Door, Window, Speaker	60
Odroid3	4	512	800		Door, Window	25
Huawei0	8	3072	12800	Camera, CO_2, Temp	Speaker	239
Huawei1	8	3072	12800	Camera, CO_2, Temp	Door, Window, Speaker	278
Macchiato0	4	16384	800			519
Macchiato1	4	4096	800			389

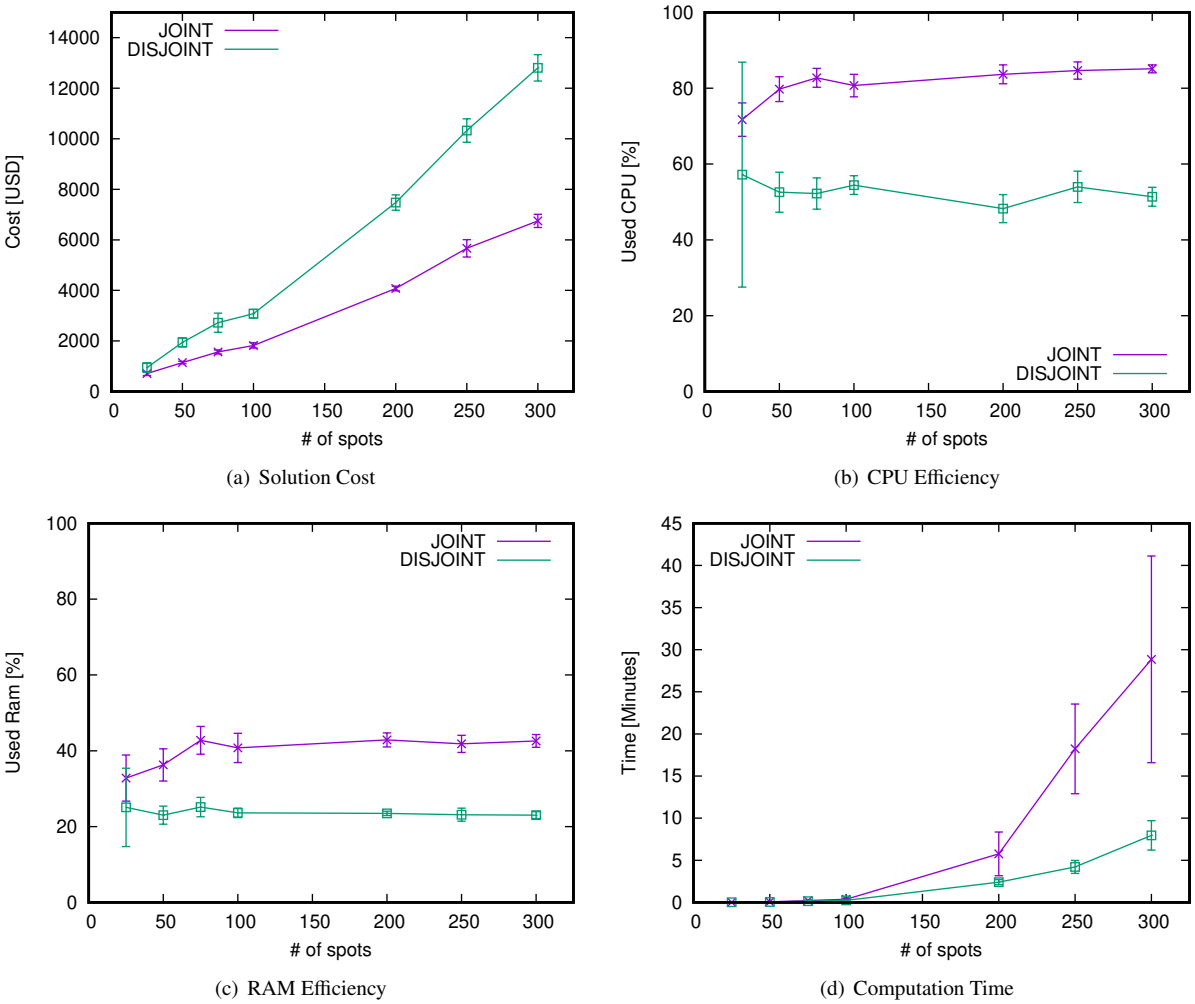
We applied the proposed approach, here dubbed as *JOINT*, to this case study and we compared the obtained solution against the solution obtained by a basic approach, called *DISJOINT* that considers all zones as separate and optimizing the instantiation of the processing node(s) in each zone as an independent problem, i.e. not considering the planimetry in the design process.

The solutions obtained by the two approaches are depicted in Figures 3(b) and 3(c) respectively, where the used spots are represented as black circles. Our solution is composed of 12 nodes while the basic one of 17 nodes, and they present relevantly different costs: 610USD and 1055USD respectively, i.e. obtaining a 42% monetary saving. We accurately investigated the characteristics of such solutions to understand the motivation of such a large cost difference and we found out that there are two main reasons. The more obvious one is the difference in the number of instantiated nodes, due to the possibility enabled by our approach to share the processing resources of the various nodes among different zones; in particular, our solution exploited the 83% of the CPU (that is the most required processing resource in the defined case study) on average, while the basic solution only the 60%.

Moreover, we found out that there is a significant difference in the chosen types of boards. Since both fire detection and video intrusion surveillance have to be executed in every zone, the independent optimization of each zone performed by the basic approach often selects a more expensive board type hosting all the required sensors and actuators. On the other hand, our approach performs a global optimization that allows to avoid overprovisioning not only of the processing power but also of the necessity of sensors and actuators. As a consequence, our approach is able to instantiate cheaper boards providing only a subset of the required sensors and actuators to be shared by the tasks executed in different overlapping zones. In the case study, the basic solution features 90 sensors and actuators integrated in the various boards among which only 70 ones are actually used (77%); on the other side, in our solutions, the architecture contains only 53 sensors and actuators among which 44 ones are used (83%).

6.2. Performance Evaluation

In order to perform a systematic evaluation of the approach, we implemented an automated strategy to formulate pseudo-realistic synthetic problems that, based on the required number of zones and spots, generates



(a) Solution Cost

(b) CPU Efficiency

(c) RAM Efficiency

(d) Computation Time

Figure 5: Performance evaluation.

a planimetry and the required set of tasks. Indeed, the board repository is the same presented in Section 6 as well as the task graph which is the one reported in Figure 4. Differently from the previous case, we allow spots with three different communication technologies: WiFi, Ethernet 100BASE-T and Ethernet 1000BASE-T.

Task instances are randomly generated and placed in each zone with different probabilities. An iterative process is performed, each zone is scanned sequentially and, for each zone, a one task instance of each different task type is added according to its specific probability. Specifically, the tasks used to collect data from CO_2 and temperature sensors and the alarm tasks are deployed in a zone with a probability of 75%. The camera acquisition task has an associated probability equal to 50%, whereas the door and window related tasks (are less frequently deployed with a probability of 30%.

The motion detection tasks have been considered zone-independent tasks, in fact, a motion detection task can potentially run in any zone since it only performs data manipulation over streams arriving from cameras, i.e. no attached sensors. However, in order to generate a realistic problem, we added multiple motion detection tasks following an approach similar to the one implemented above; specifically, for each zone we included this additional type of task with a probability equal to 40%. Finally, we also introduced another type of task that combines camera acquisition with motion detection; As for motion detection, it is zone-independent because the task already has all the needed sensors in place, i.e. each task combines the camera stream and the motion detection algorithm; therefore, no specific

zone placement is required. The probability of adding this task is 30% for each zone.

Regarding the computation requirements, each task consumes a certain amount of CPU, RAM and disk according to its task type and the selected board type. Specifically, RAM and disk requirements have been set considering only the task type because they are homogeneous across different boards (final values between 5 and 250 for RAM and between 1 and 20 for disk). The CPU requirements, instead, have been randomly chosen based on the task type and on the performance table, resulting in a distribution between 0.35 and 2 cores for each task.

In order to steer the problem generation algorithm, we also introduced three additional parameters: i) *maximum spots per zone*, ii) *maximum zones per spot*, and iii) *maximum connected zones*. The first and the second parameters are used to define the density of spots and, to some extent, the maximum number of overlapping zones. The third parameter, instead, is exploited to take into account zone-independent tasks. Specifically, even if such tasks can be deployed almost in any zone w.r.t. spot's bandwidth limitations, we tried to generate problems that can reflect real environments. To this aim, we assumed that zone-independent tasks must be deployed within a certain area around the source and target tasks. In other words, we aggregated close zones in a sort of clusters, called areas, and we assume that zone-independent tasks must be deployed within the area of their source and target tasks. If this is unfeasible, additional source or target tasks are added to the problem. The *maximum connected zones* parameter defines the maximum size of the cluster.

Again, even in this case, we compared our *JOINT* solution against the *DISJOINT* baseline, where the concept of zones overlapping and consequent spot sharing are not exploited. To perform an extensive analysis, we ran a set of experiments by varying the number of spots (from 25 to 300) and consequently the number of zones (from 15 to 150, approximately). The obtained results are reported in Figure 5, where each point represents the average of 10 different problems of a specific size and it is reported along with the 95% confidence interval. Moreover, to limit execution times for both approaches, we halt the MILP solver as soon as the obtained solution is proven to differ from the optimal solution for less than 5%.

The solution cost is reported in Figure 5(a); as can be seen, the overall cost is proportional to the number of spots in both the two cases. Indeed, the number of spots increases the problem dimension and therefore the number of boards that must be instantiated. However, it is worth to note that, even with small problems characterized by 25 spots, the *JOINT* approach finds a solution with a cost around 700 on average, whereas the *DISJOINT* approach solves the problem with an overall cost of 945 on average. With larger problems the difference between the two approaches increases significantly. For instance, with 250 spots the *DISJOINT* approach finds a solution that costs almost double the solution found by our *JOINT* approach.

We also analyze the CPU and RAM usages of the obtained solutions, in Figure 5(b) and 5(c), respectively. Disk usage presents results similar to the ones of CPU and RAM usages; thus such results have not been reported for the sake of space. As can be seen the *JOINT* approach can, on average, use the 80% of the CPU and the 40% of RAM in all configurations, whereas the *DISJOINT* achieves the 50% and the 20 %, respectively. As in the previous use case, this confirms that the *JOINT* approach assumes that some spots are shared between zones and therefore tasks from different zones can be allocated to the same board deployed in the shared spot.

The side effect of considering the planimetry in our *JOINT* approach is a higher complexity in the problem in input to the MILP solver. We report in Figure 5(d) the statistics on the execution times needed by the solver to obtain a valid solution for each problem. With small and medium size problems (with less than 200 spots) the performances are equal and the computation time is negligible. With more spots, instead, the performance of the two approaches differ significantly. The *DISJOINT* approach can find a solution in less than 10 minutes in all configurations, whereas in the *JOINT* case, the MILP solver can require up to 40 minutes to find a solution proven to be 5% far from the optimal one. Moreover the graph shows that execution time grows in an over-linear way w.r.t. the problem size. However, it is important to remark that the synthesis problem analyzed in this work should be run offline only one time during the design process (or only when the workload changes); therefore a solving time of less than one hour could be acceptable in most cases. Moreover, scenarios with 300 spots are particularly huge and not so frequent in real use cases.

7. Conclusions

We have presented a methodology for the automated design and deployment of distributed cyber-physical systems in smart environments. The proposed approach takes into account a realistic environmental model presenting overlapping geographical locations for cyber-physical interactions of different applications, and per-

forms a joint co-exploration of the architecture instantiation and of the corresponding applications mapping by means of a MILP optimization engine. We proved the effectiveness of our approach to a smart office case study by comparing the obtained solution with the one produced by a basic reference approach that optimizes each zone in the environment individually. Our solution employs fewer boards than the basic solution also optimizing resource utilization. Moreover, our solution is able to select cheaper boards, still able to satisfy the applications' requirement, resulting in a 42% cost saving. An extensive experimental campaign using synthetic problems has confirmed such results; here, the cost saving is up to the 50% for large size problems; at the same time execution times of the optimization process showed to be affordable within the design-time process. Future work will focus on extending the proposed methodology to consider fault tolerance and security issues.

Acknowledgments

This work was partially supported by the Italian Ministry of Education and Research in the framework of the CrossLab Project, Departments of Excellence.

References

[1] ABB. Industrial IoT Applications. `https://new.abb.com/control-systems/features/industrial-IoT-services-people-use-cases`. Accessed: 2019-07-20.

[2] S. Ahn, M. Gorlatova, P. Naghizadeh, and M. Chiang. Personalized Augmented Reality via Fog-Based Imitation Learning. In *Proceedings of the Workshop on Fog Computing and the IoT (IoT-Fog)*, IoT-Fog '19, pages 11—-15. ACM, 2019.

[3] M. Al-khafajiy, H. Kolivand, T. Baker, D. Tully, and A. Waraich. Smart hospital emergency system. *Multimedia Tools and Applications*, 78(14):20087–20111, Jul 2019.

[4] M. S. Ardekani, R. P. Singh, N. Agrawal, D. B. Terry, and R. O. Suminto. Rivulet: A Fault-tolerant Platform for Smart-home Applications. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 41–54. ACM, 2017.

[5] A. Bakshi and V. K. Prasanna. Algorithm design and synthesis for wireless sensor networks. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 423–430 vol.1. IEEE Computer Society, 2004.

[6] N. Bombieri, F. Fummi, and D. Quaglia. System/Network Design-space Exploration Based on TLM for Networked Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 9(4):37:1–37:32, Apr. 2010.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceeding of Workshop on Mobile Cloud Computing (MCC)*, pages 13–16. ACM, 2012.

[8] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu. Fog Computing: A Platform for Internet of Things and Analytics. In N. Bessis and C. Dobre, editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[9] A. Brogi and S. Forti. QoS-Aware Deployment of IoT Applications Through the Fog. *IEEE Internet of Things Journal*, 4(5): 1185–1192, Oct 2017.

[10] E. Ebeid, F. Fummi, and D. Quaglia. Model-Driven Design of Network Aspects of Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(4):603–614, April 2015.

[11] E. Fraccaroli, F. Stefanni, R. Rizzi, D. Quaglia, and F. Fummi. Network Synthesis for Distributed Embedded Systems. *IEEE Transactions on Computers*, 67(9):1315–1330, Sept 2018.

[12] Huawey. Edge Computing IoT (EC-IoT) Solution. `https://e.huawei.com/en/solutions/business-needs/enterprise-network/agile-iot`. Accessed: 2019-07-20.

[13] A. J. Hussain, D. M. Marcinonyte, F. I. Iqbal, H. Tawfik, T. Baker, and D. Al-Jumeily. Smart Home Systems Security. In *Proceedings of IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1422–1428. IEEE, 2018.

[14] IBM Corporation. CPLEX Optimizer. `https://www.ibm.com/analytics/cplex-optimizer`. Accessed: 2019-09-16.

[15] M. T. Lazarescu. Design of a WSN Platform for Long-Term Environmental Monitoring for IoT Applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1): 45–54, March 2013.

[16] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., USA, 1990. ISBN 0471924202.

[17] M. Mukherjee, L. Shu, and D. Wang. Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges. *IEEE Communications Surveys Tutorials*, 20(3):1826–1857, thirdquarter 2018.

[18] M. Naas, L. Lemarchand, J. Boukhobza, and P. Raipin. A Graph Partitioning-based Heuristic for Runtime IoT Data Placement Strategies in a Fog Infrastructure. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC)*, pages 767–774. ACM, 2018.

[19] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu. Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets. *IEEE Internet of Things Journal*, 4(5):1216–1228, Oct 2017.

[20] OpenFog. OpenFog Use Cases. `https://www.openfogconsortium.org/resources/#use-cases`. Accessed: 2019-07-20.

[21] Oracle Corporation. Industry 4.0 with Oracle IoT. `https://www.oracle.com/it/internet-of-things/solutions.html`. Accessed: 2019-07-20.

[22] A. Puggelli, M. M. R. Mozumdar, L. Lavagno, and A. L. Sangiovanni-Vincentelli. Routing-Aware Design of Indoor Wireless Sensor Networks Using an Interactive Tool. *IEEE Systems Journal*, 9(3):714–727, Sept 2015.

[23] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana. Fog Computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology*, 19(2):18:1–18:41, Apr. 2019.

[24] P. Sayyah, M. Lazarescu, S. Bocchio, E. Ebeid, G. Palermo, D. Quaglia, A. Rosti, and L. Lavagno. Virtual Platform-Based Design Space Exploration of Power-Efficient Distributed Embedded Applications. *ACM Transactions on Embedded Computing Systems*, 14(3):49:1–49:25, Apr. 2015.

[25] C. Shen, R. P. Singh, A. Phanishayee, A. Kansal, and R. Ma-

hajan. Beam: Ending Monolithic Applications for Connected Devices. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC 16)*, pages 143–157. USENIX Association, 2016.

[26] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner. Optimized IoT Service Placement in the Fog. *Service Oriented Computing and Applications*, 11(4):427–443, Dec 2017.

[27] E. Sturzinger, M. Tornatore, and B. Mukherjee. Application-aware resource provisioning in a heterogeneous Internet of Things. In *Proceedings of the International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–6. IEEE, 2017.

[28] M. Taneja and A. Davy. Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In *Proceedingd of the IFIP/IEEE Sympodium on Integrated Network and Service Management (IM)*, pages 1222–1228. IEEE, 2017.

[29] S. Xu, R. Kumar, and A. Pinto. Correct-by-Construction and Optimal Synthesis of Beacon-Enabled ZigBee Network. *IEEE Transactions on Automation Science and Engineering*, 10(1): 137–144, 2013.

[30] S. Yi, C. Li, and Q. Li. A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the Workshop on Mobile Big Data (Mobidata)*, pages 37–42. ACM, 2015.

[31] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li. LAVEA: Latency-Aware Video Analytics on Edge Computing Platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*, pages 2573—-2574. ACM, 2017.

[32] L. Yin, J. Luo, and H. Luo. Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacture. *IEEE Transactions on Industrial Informatics*, 14 (10):4712–4721, Oct 2018.

[33] L. Zhao and J. Liu. Optimal Placement of Virtual Machines for Supporting Multiple Applications in Mobile Edge Networks. *IEEE Transactions on Vehicular Technology*, 67(7):6533–6545, July 2018.

[34] Z. Zhou, J. Hu, Q. Liu, P. Lou, J. Yan, and W. Li. Fog Computing-Based Cyber-Physical Machine Tool System. *IEEE Access*, 6:44580–44590, 2018.