The Journal of Systems & Software 183 (2022) 111086

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Supporting IoT applications deployment on edge-based infrastructures using multi-layer feature models ☆

Angel Cañete *, Mercedes Amor, Lidia Fuentes

*Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain*
*ITIS Software, Universidad de Málaga, Spain*

ARTICLE INFO

*Article history:*
Received 10 January 2021
Received in revised form 12 May 2021
Accepted 2 September 2021
Available online 16 September 2021

*Keywords:*
Task allocation problem
Variability models
Task Deployment
SMT Optimization
Software Product Lines
Edge Computing
DevOps

ABSTRACT

Edge Computing proposes to use the nearby devices in the frontier/Edge of the access network for deploying application tasks of IoT-based systems. However, the functionality of such cyber–physical systems, which is usually distributed in several devices and computers, imposes specific requirements on the infrastructure to run properly. The evolution of an application to meet new user requirements and the high diversity of hardware and software technologies in the IoT/Edge/Cloud can complicate the deployment of continuously evolving applications. The aim of our approach is to apply Multi Layer Feature Models, which capture the variability of applications and the software and hardware infrastructure, to support the deployment in edge-based environments of cyber–physical applications. With this multi-layered approach is possible to support the evolution of application and infrastructure independently. Considering that IoT/Edge/Cloud infrastructures are usually shared by many applications, the deployment process has to assure that there will be enough resources for all of them, informing developers about the feasible alternatives. We provide four modules so that the developer can calculate what is the configuration of minimal set of devices supporting application requirements of the evolved application. In addition, the developer can find what is the application configuration that can be hosted in the current infrastructure. The successive solutions of continuous deployment generated by our approach pursue the reduction of the system energy footprint and/or execution latency.



## 1. Introduction

IoT/CPSs (cyber–physical systems) are a new generation of software intensive systems that are composed by software tasks with specific needs in terms of hardware and software. The functionality of CPSs needs to be deployed throughout servers in the Cloud, and the system infrastructure, which is typically composed of a myriad of highly heterogeneous devices that differ in their type, memory, computation power, coupled sensing units, operating system or communication capabilities among others. Cloud servers provide high performance computational resources that can be used to offload processing and storage resources alleviating resource-constrained devices (Ai et al., 2018; Premsankar et al., 2018; Satyanarayanan, 2017). However, cloud offloading pays a heavy cost in terms of energy and latency (Bulej et al., 2021; Ai et al., 2018; Shi et al., 2019). With the goal of reducing the energy footprint of IoT/CPSs, novel technologies and paradigms based on Edge Computing (EC) (Premsankar et al., 2018; Elazhary, 2018) propose to offload some computation intensive tasks from the cloud servers onto nearby edge devices located in a range of one or two hops (e.g. routers or switches). Edge-based deployments take advantage of the inactive computational capacity and unused storage space of edge devices placed at the Internet's frontier. Then, part of the data processing and storage will be displaced to devices closer to where data and services are produced and consumed, proposing a more sustainable solution.

However, despite its advantages, the realization of edge-based deployments is not, in practical terms, an easy issue with current approaches (Bagchi et al., 2020; Mao et al., 2017; Liu et al., 2019; Huang et al., 2020). Roughly, the deployment process of IoT/CPSs can be formulated as a task assignment problem that has an miscellaneous set of tasks with variable requirements in infrastructure and QoS (quality of service) in one side, and an elastic set of heterogeneous devices in the edge or the cloud in the other side. However, the heterogeneity of existing edge-infrastructures and the requirements imposed by applications complicate the deployment process, making the developer to face some challenges. On the one hand, the high diversity of hardware

☆ Editor: W.K. Chan.

* Correspondence to: Escuela Técnica Superior de Ingeniería Informática, Bulevar Louis Pasteur, 35, Málaga, 29010, Spain.
*E-mail addresses:* angelcv@lcc.uma.es (A. Cañete), pinilla@lcc.uma.es (M. Amor), lff@lcc.uma.es (L. Fuentes).

https://doi.org/10.1016/j.jss.2021.111086

and software technologies in the Edge makes it more difficult to find out what is the best deployment as IoT/CPS applications evolve over time—due to changes in user requirements that may require application functionality adjustment (Bagchi et al., 2020). On the other hand, it is also hard to discover when would be advantageous to offload some task(s) to a certain Edge device as resource availability varies over time.

First, before making any offloading decision, it is necessary to know if the infrastructure in the Edge meets application specific requirements. The problem is that some tasks are constrained to a concrete node with specific software and hardware resources. Examples of these requirements are that a certain node must include a camera, a special sensing unit, specific support for a virtual machine, or particularly high performance computational resources. In some cases, the software engineer must reuse a (pre-existing) given infrastructure, and the application functionality has to be adapted to the capabilities and resources provided by the available infrastructure. In order to prevent the developer to configure applications that will not be supported by the infrastructure at their disposal, first it must be checked if the current infrastructure supports the desired application configuration. In other cases, the software engineer is free to acquire the most appropriate infrastructure for a CPS that meets application requirements. In this scenario, the goal is to identify and configure what is the minimal set of devices that should be included to deploy and run a given application configuration.

Then, the next step is to decide how the application can be allocated in an IoT/Edge/Cloud environment. This decision should consider the task latency and also the computational and communication power consumption, and find different offloading solutions depending on each task resource demand and its delay sensitivity (Mao et al., 2017; Liu et al., 2019; Huang et al., 2020). Among these solutions, it is preferable to choose those that reduce energy consumption, to contribute producing more sustainable applications while minimizing latency to provide a good quality of experience to users. So, considering that infrastructures are commonly shared among many applications, the next goal is finding what subsets of devices are capable of running an application configuration. After that, it is necessary to determine what subset of devices should be selected in order to minimize QoS, such as the energy consumption and/or latency, and providing a task assignment to a certain infrastructure, fulfilling a certain QoS.

We propose to tackle this deployment problem using a software product line (SPL) (Pohl et al., 2005) approach, which provides the means for the identification, representation, and traceability of variability in many domains (Geraldi et al., 2020). We model the variability of software tasks and infrastructure by means of variability models, concretely multi-stage (i.e., multi-layer or multi-view) feature models (FMs) (Dhungana et al., 2010; Reiser and Weber, 2007; Holl et al., 2012; Acher et al., 2012, 2013). The definition of different models allows to specify the variable set of hardware and software characteristics of the devices independently from the application. This supports the automation of the configuration of different software infrastructures (e.g. a virtual machine or an IoT platform) to make possible that a device can host a certain application task.

These multi-stage feature models are the input of four modules that focus on providing an automated solution to these challenges. Specifically, the developer can use these modules to (1) adapt the application configuration according to the capabilities of the infrastructure. If it is not possible to find a solution to adapt the application configuration to an existing infrastructure, the module suggest new devices and another module can (2) characterize them—in terms of required resources. Once the developer is ascertain about deployment is possible, another module (3) finds out what subsets of devices are candidate to be part of the deployment solution and where to deploy each application's task. After that, module (4) determines what subset of devices should be selected in order to minimize QoS, such as the energy consumption and/or latency, and provides a task assignment to a certain infrastructure that guarantees a QoS. These modules are able to manage the configuration of both the application and the infrastructure independently. Our approach provides a mechanism based on variability models that defines and optimizes (in terms of energy consumption and/or latency) the deployment of IoT/CPSs applications for a specific edge infrastructure. This work aims to aid developers to take advantage of Edge devices' unused computational capacities during the operation of IoT/CPSs applications to ensure a successful deployment. Several approaches deal with the automated assignment of application components to infrastructures composed of several nodes (Mao et al., 2017; Liu et al., 2019), but these works neither consider the software characteristics of the nodes, nor identify what are the application required features unsupported by the nodes. So, unlike our approach, these solutions may lead to unfeasible deployments in which some tasks cannot be executed since an Edge device does not provide the required software or hardware support (e.g. a camera).

The rest of the paper is organized as follows: Section 2 discusses related work, while Section 3 presents our approach, including our four reasoning modules in depth. Section 4 presents two extensible and reusable FMs for IoT infrastructures and their constraints. Section 5 applies our approach to a real scenario of a college campus, and evaluates the benefits obtained and its scalability. Finally, Section 6 concludes the paper and presents the future work.

## 2. Related work

This section discusses the related work on variability management and code offloading.

### 2.1. Variability management: SPL and IoT

The use of SPL has being proved to provide benefits to IoT systems regarding the management and modeling of variability, which is mainly managed by FMs with variation points, variants, relationships and constraints (Geraldi et al., 2020). Just a few works address variability at the deployment stage taking into account features that model the diversity of existing devices and technologies into consideration (Gámez and Fuentes, 2013). The work in Cecchinel et al. (2016) applies SPL concepts to manage sensor platform variability problem in terms of memory, controller, battery, sensor and communication. The work in Köksal and Tekinerdogan (2019) applies SPL to manage the variability in IoT systems in agriculture, which typically have different functional and quality requirements such as choice of communication protocols, the data processing capacity, the security level, safety level, and time performance.

Despite its importance, the infrastructure is often neglected in the SPL models. Typically, when included, the modeling of infrastructure-specific features is intermingled with the rest of application-specific features, mapping the application's features to pieces of software in a 1:1 relation. Separating the platform and infrastructure-dependent feature into an independent model aims of reflecting the possible independent evolution and the restrictions that the infrastructure imposes on the deployment of software applications. The definition of separated but related FMs is not new, as maintaining a single large FM for an entire system is neither feasible nor desirable. Several works (Dhungana et al., 2010; Reiser and Weber, 2007; Holl et al., 2012; Acher et al., 2012, 2013) propose defining multiple FMs to cope with variability management of large and complex FMs related by

numerous complex constraints. Each FM groups features representing different viewpoints, sub-systems, concerns of the software system (Acher et al., 2013), or levels of abstraction, such as platform descriptions (Lettner et al., 2019; Farahani and Habibi, 2019), or stages of development (Rabiser et al., 2016).

The framework presented in Lettner et al. (2019) uses two layers (by multi-layered FMs) to develop applications executed in mobile phones. Those device's features whose changes strongly affects the application configurations dependent on the underlying platform are separated in a FM. The work in Farahani and Habibi (2019) proposes to separate the modeling application and infrastructure in two FMs.

Although the separated modeling of infrastructure features promotes its independence and reusability across different application domains, these approaches just allow the configuration of a single device, preventing its application to infrastructures composed of multiple nodes. Consequently, they are not suitable for being applied to real EC environments. This neglect in relation to modeling platform dependencies also limits software infrastructure optimization, as many features (e.g., performance, energy consumption) depend on the devices in which applications run (Abbas et al., 2018; Guo et al., 2011). In addition, none of the aforementioned approaches helps the developer to know if a configured infrastructure does not meet the configured application functionalities, or how to resolve the match between the application demand to infrastructure support. Also, the way non-functional requirements are modeled limits the reusability of the infrastructure's FM for different applications, and does not consider the coexistence of several applications running simultaneously in a sharing an infrastructure and difficulties the FM maintenance and evolution of the application and/or the infrastructure. Also, only contemplate hardware infrastructure features, but nowadays, with the virtualization options and the proliferation of IoT middleware, the software variability in cyber–physical systems is getting much importance.

In contrast, our approach supports the explicit modeling of heterogeneous edge nodes, IoT devices and cloud machines, which differ in both software and hardware resources. This model, which can be extended and reused, allows reasoning about the constraints and interrelationships among application requirements, software infrastructure and hardware. Our approach also allows the management of alternative implementations for the same task interface. The possibility of having more than one implementation of a task allows finding out the best tasks' implementation for the different nodes of the infrastructure, considering a certain quality of service (e.g. low energy consumption). We also consider the real scenario in which the deployment nodes (i.e. the infrastructure) are often shared among a set of applications, so both software infrastructure and resources available are variable and not fixed as in many other proposals. Finally, our approach provides a customizable optimizer of edge-based application deployments with the goal of minimizing energy consumption, which allows considering the energy in both, the whole application and isolated nodes (customizable according to the necessities of the developer/infrastructure). In this way, we provide very flexible task allocation solutions.

### 2.2. Task assignment problem: Offloading

On the other hand, our approach is also focused on task offloading in edge infrastructure. Existing task offloading approaches differ with respect to the type of edge nodes considered for offloading and the number of user considered. Most of the approaches contemplate only one edge node (or a group of them with homogeneous characteristics), being focused on systems formed by one mobile user (single user) or multiple mobile nodes (multiuser approaches) (Mao et al., 2017; Liu et al., 2019; Huang et al., 2020). Considering only homogeneous nodes, as these approaches make, allow to reduce the complexity of task allocation to just deciding if the computational load is allocated locally or remotely. A more realistic but complex scenario, involving a heterogeneous set of edge nodes (in hardware, software and available resources) shared among multiple users, requires deciding which node is the most adequate for offloading.

Task offloading approaches also differ in the objective(s) that drive it (e.g., reducing the energy consumption, latency, etc.), the nodes involved (user node, all nodes, etc.), granularity level, task model used, mechanisms used to obtain the solution, and devices' characteristics considered (Mao et al., 2017; Cañete et al., 2020). The granularity level indicates the size of the computational load that is considered for offloading (tasks, services, or applications), while the task model determines the use of offloading scheme, mainly binary or partial offloading (Mao et al., 2017); the binary offloading does not allow the partition of tasks, while the partial offloading allows splitting tasks into simpler components, providing more flexible solutions. Most of the approaches dealing with the tasks assignment problem in code offloading use heuristic algorithms or solve this problem by means of Integer Linear Programming (ILP), Mixed Integer Linear/Non-linear Programming (MILP/MINLP) and SMT (Satisfiability Modulo Theories) solvers (Mao et al., 2017). While heuristic algorithms do not ensure to return a solution (even if exists), ILP, MILP/MINLP and SMT solvers always return a solution if it exists, or on the contrary inform about the infeasibility of the problem. ILP and MILP/MINLP solvers differ in the type of variables and degree of the equations supported: in ILP, all variables are restricted to take integer values and equations must be linear; MILP and MINLP can support other types of variables, but at least a variable must be integer-valued, being the equations linear/non-linear respectively. Finally, SMT solvers do no restrict the type of variables to be optimized and support non-linear equations. Our approach considers heterogeneous nodes and multiples simultaneous users, allows to minimize the energy consumption and/or latency and customize where it is focused, works at tasks level, allows partial offloading, contemplates software and hardware characteristics of the nodes and uses a SMT solver to provide an optimal solution.

As far as we know, only our previous work (Cañete et al., 2020) considers software characteristics of the infrastructure (such as operating system, support for software virtualization, or third party libraries) when deciding the tasks allocation (Mao et al., 2017; Liu et al., 2019; Cañete et al., 2020). Nevertheless, while our previous work did not consider optimizing the latency in task assignment solution, this work does it. Compared to the rest of approaches, our approach allows more fine-grained offloading scheme than current binary offloading, which allows supporting parallel execution between the user mobile and the edge nodes. Additionally, our approach provide a mechanism to set the optimization according to the user/infrastructure necessities, while the rest have this parameter fixed.

## 3. Our approach

In this work we present four independent but complementary modules that supports developers to adjust the deployment according to the application requirements and infrastructure capabilities: the Application Variability Adaptor (AVA), the New Devices Finder (NDF), the Edge-Deployment Alternatives Finder (EDAF), and the Energy and Latency Minimizer (ECLAM). The relation of these modules with others artifacts of our approach is described in Fig. 1.

Firstly, the software engineer is concerned with the definition of the FMs of the application and infrastructure. A FM represents,

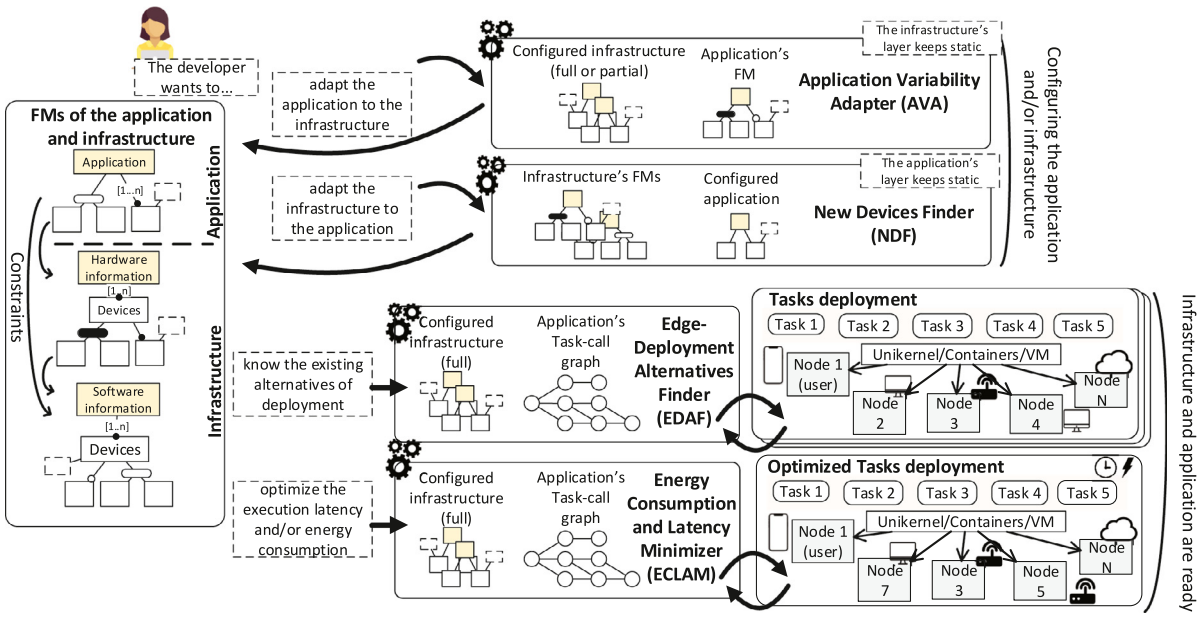


**Fig. 1.** Full overview of our approach.

in terms of features, which elements of a family of products (either a family of applications or a family of systems) are common, which are variable and the relationship between them. FMs are represented as a set of hierarchically ordered features, composed of parent–child relationships and a set of constraints (called *cross-tree constraints*) which represents the relationships among features.

Infrastructures are formed of several heterogeneous devices, described by a set of hardware and software characteristics: type of device, computing power, peripherals, network capabilities, operating system, third party libraries, etc. These characteristics relate to the type of services/tasks that can be deployed on them, as they refer to the hardware and software requirements demanded by the applications to the execution platform. This FM allow the modeling more adequately the infrastructure, often neglected in the SPL models, and allows mapping the application's features selected from the application FM to features of software and hardware of the infrastructure in a 1:n relations (an application feature can be executed by nodes with different features). To facilitate the configuration and reusability of the infrastructure's FM, we split the hardware and software features in two FMs, which aids us to manage the evolution of the hardware and software characteristics of the devices separately; i.e., we use three FMs in total, one for the application and two to model the infrastructure.

The information provided by these FMs is used by the modules to:

***(1) Adapt the variability of the application according to the capabilities of the infrastructure (Application Variability Adaptor—AVA module):*** having a concrete infrastructure's configuration (either partial or complete), this module returns the application configuration with the maximum number of application's features that fits a given infrastructure. In this scenario, we want to provide the most functionality allowed by the current infrastructure as possible.

Sometimes, the application must provide a specific set of services, being the infrastructure that must be adapted, i.e., is the application layer that keeps static. That means that the infrastructure may require new devices to support the desired application.

***(2) Add new devices to the infrastructure (New Devices Finder—NDF module):*** this module informs about the new devices needed to run a specific application. Additionally, the evolution of the application and/or infrastructure may affect the feasibility of the current application's deployment. Some tasks or parts of the application may require devices with specific characteristics (in terms of location, peripherals, computation capabilities, etc.) that the current infrastructure does not meet. This module detects all application's requisites non-supported by the current configuration of the infrastructure, and returns a minimal configuration (i.e., the minimal set of features and minimal value of the attributes) of the minimal set of devices needed to deploy the application properly.

Once the software product and the infrastructure are configured, the next step is to distribute the application's tasks among the nodes of the infrastructure (see Fig. 1). Such distribution requires to ***(3) find the deployments alternatives capable of supporting the application execution (Edge-Deployment Alternatives Finder—EDAF module):*** this module informs about the task-to-device assignments that satisfy the requirements of the application, including the amount of memory (in terms of RAM—Random Access Memory and disk) required to allocate in each computing device and the maximal number of users each deployment solution would support—allowing to set this last value to return solutions with a minimal number of users supported. If the developer unknowns if the infrastructure is capable of executing the application properly, the EDAF could not be able to find any deployment solution. Although the NDF and the EDAF are independent between them, the previous execution of the NDF module (and the posterior purchase of the required devices found, which are not in the current infrastructure) assures the accomplishment of the application requirements by the infrastructure, and, consequently, to find at least one solution of deployment (see Fig. 1).

While all the alternatives of tasks distribution provided by the EDAF fulfill the application's requirements, this module does not give information about the energy consumption and latency of the deployment solutions provided.

***(4) Optimize the deployment by minimizing the latency and/or energy consumption (Energy Consumption and Latency Minimizer—ECLAM module):*** configurations returned by module (3) meet the requirements of the application, but they may differ in the energy consumption and execution latency (apart from the

number of users supported), which depend on the characteristics of the devices and tasks deployed in each of them (Mao et al., 2017). The goal of this module is to evaluate these parameters (i.e. energy consumption, execution latency and user supported) in the deployment solutions, according to the computation and code offloading demands and the capabilities of the devices, and minimizes one or more of them (De Moura and Bjørner, 2008). As a result, this module returns only one of the solutions found by the EDAF, the optimal one. In addition, this module allows to customize the devices in which the reduction in the energy consumption is focused; e.g., just on the user device, regardless the rest of devices (so this policy will offload as much functionalities as possible from the user device), on battery powered devices, etc. Further details are provided later in Sections 4 and 3.4. Like the EDAF, this module allows setting the number of users supported, which guarantees that the deployment solution will be able to serve a certain number of simultaneous users. Although the tasks assignment returned by this module is among the found by the EDAF, both modules are independent; i.e., can be used separately.

The four modules are posed as constraint satisfaction problems. Concretely, we rely on Z3, a SMT solver (De Moura and Bjørner, 2008), to obtain the solutions. Thus, (1) the algorithm of the solver always returns a solution (unlike heuristic algorithms), which guarantees that the deployment is feasible or the impossibility to deploy the application if no solution is found; and (2) a large number of constraints (required to solve the problems at hand) help SMT-solvers to reduce the search space and to find the optimal solution faster (Niewiadomski et al., 2014; Bjørner et al., 2015). Unlike solvers that use ILP/MILP/MINLP (see Section 2.2), SMT solvers neither restrict the variables types involved in the problem formulation (that are unknown, as our modules inputs are derived from the FMs) nor the degree of the equations. Additionally, it is proved that Z3, and concretely its optimization module ($\nu$Z), returns optimal solutions (Bjørner et al., 2015). Nevertheless, the flexibility of our approach may allow the use of any other mathematical model (that do no restrict the variables types) for optimization.

### 3.1. Module 1: Application Variability Adaptor (AVA)

This module is useful to foresee if a given infrastructure supports the deployment of an application. Some application's functionalities require specific hardware and/or software components to be executed. The goal is not to adding more nodes to the infrastructure, but to constraint the variability of the application (i.e. those features that are optional or adaptable) to the infrastructure capabilities and resources. The aim of the AVA is to find out whether the application's features are supported by a given infrastructure.

To operate with the FM of the application, it is transformed in a tuple $\langle F, A, R, CMCs\rangle$, where:

- $F$ contains the set of boolean features included in the model. In case of configurations, F contains the selected features.
- $A$ is the set of variables that represent the attributes of the model, which value is a real number.
- $R$ identifies the set of clonable services of the application (i.e., those that can be instantiated more than once). At the same time, each clonable service included in R is formed by tuples $\langle F, A\rangle$, that contain the children features and attributes of each clonable service. Clonable services are typically executed on devices located on a specific location.
- *CMCs* (Cross-Models Constraints) represents the set of constraints between models. According to the model, it will be related with none, one or more models (see Fig. 1).

The main operation of this module uses the next additional functions:

- *support(device(s), feature/attribute, restrictions)*: its value is *True* if device(s) support(s) *device(s)/attribute(s)* according to *restrictions*; *False* in other case.
- *capabilities(clonable, device, restrictions)*: returns the limitations that *device* imposes in *clonable* in case of being assigned this feature to it, according to *restrictions*.

As an input, the AVA module receives the application's FM, as well as the hardware and software configuration of the infrastructure (either full or partial), split into: $N$, that contains the nodes; and *CMCs*, that contains the hardware and software restrictions between the infrastructure and the application. As output, the AVA returns the features split into two groups: supported ($s$) and non-supported features ($ns$) by the infrastructure. Regarding the attributes, the AVA module returns their value range in case of being restricted by the infrastructure.

$$
\begin{aligned}
&\textbf{Maximize}: && size(s) && \\
& && \forall a \epsilon A : a_{max} && \\
&\textbf{Minimize}: && \forall a \epsilon A : a_{min} && \\
&\textbf{Subject to}: && \forall f \epsilon s : support(N, f, CMCs) && (1)\\
& && \forall a \epsilon A : support(N, a_{max}, CMCs) && (2)\\
& && \forall a \epsilon A : support(N, a_{min}, CMCs) && (3)\\
& && \forall n \epsilon N, c \epsilon R : c \epsilon S && \\
& && \quad \wedge support(n, c, CMCs) \rightarrow && \\
& && \quad capabilities(c, d, CMCs) && (4)\\
&\textbf{Return}: && s && \\
& && ns = F \setminus s && \\
& && \forall a \epsilon A : (a_{min}, a_{max}) &&
\end{aligned}
\tag{1}
$$

Eq. (1) shows the formalization of AVA. Concretely, this module maximizes the number of supported features and upper range of the attributes, while minimizing their lower range, and maximizes the number of supported features. As constraints, the AVA: (1) checks that the infrastructure supports the features contained in *supported*; (2–3) confirms that the infrastructure supports the maximal and minimal values of each attribute; (4) iterates each clonable service and device, evaluating and returning the maximal capabilities supported by each device in case of executing the service on it. Finally, the AVA returns the supported and non-supported features, as well as the value range of the attributes.

### 3.2. Module 2: New Devices Finder (NDF)

Sometimes, the developer knows the specific application he/she wants to run on the infrastructure, but ignore which are the specific features of the device/s to do so. Starting from a specific (configured) application and a infrastructure that can be evolved to meet (unsupported) application features, the goal of this module is to define the characteristics of the minimum set of IoT/Edge/Cloud devices that are not included in the current infrastructure, yet needed to support the application. With this purpose, the NDF module evaluates the requirements of the application services – once mapped from the FM – already met by the current infrastructure, and defines an instance of the least amount of devices required to support the unsupported ones.

This module works at service level, using the demands of the application's services not already supported to determine the characteristics of the devices needed. To assure the QoS of the application related with user experience and latency, each service has a maximal execution time associated (i.e., time restriction). The NDF module performs two different steps. In the first step, the NDF detects the non supported services and instantiates one (provisional) device for each one. In the second

step, these provisional devices are combined to construct a set of indispensable devices, while avoiding to include those that are expendable(function *merge* presented below).

This module uses several additional functions:

- *meets(service,device):* returns *True* if *device* accomplishes the *service* requirements, involving their hardware and software demands and time restrictions (considering computation and communication times).
- *createDevices(non-supported):* returns one device per each non-supported service with the minimum characteristics to meet it.
- *merge(devices,virtualization):* if $device_i$ and $device_j$ included in *devices* share location – or any of both can be located anywhere – and share its type, then their peripherals, communication capabilities, sensing units and software capabilities are combined. CPU power, RAM capacity, and disk will be either combined or equaled to the device with more resources according to the boolean value *virtualization*, which value its *True* if our intention is to reserve hardware resources (i.e., use virtual machines, see Section 4.2).

Eq. (2) formalizes the NDF. As an input, this module receives the full configuration (both software and hardware) of the nodes presented in the infrastructure, the configuration of the application, and the boolean value *virtualization*. To handle this information, the configured FMs are processed, including the nodes in the set *N*, and all the services of the application included in the set *S*. The set *S* is then composed by the software components that implement the application's functionality, which are derived from the full configuration of the application FM, including those clonable services defined as part of the set *R*. As an output, the NDF returns the minimal set of characteristics required by each new device to support the application configuration given as an input.

$$
\begin{aligned}
&\textbf{Maximize}: && size(s) && \\
&\textbf{Subject to}: && \forall se \epsilon S,\, n \epsilon N : x_{se,n} \rightarrow se \epsilon s \wedge meets(se, n) && (1)\\
& && n \epsilon N : se \epsilon S \rightarrow \sum_{se \epsilon S} x_{se,n} se_{RAM} \leq n_{RAM} && (2)\\
& && n \epsilon N : se \epsilon s \rightarrow \sum_{se \epsilon S} x_{se,n} se_{disk} \leq n_{disk} && (3)\\
&\textbf{Post processing}: && ns = F \setminus s && (4)\\
& && provisional = createDevices(ns) && (5)\\
&\textbf{Return}: && merge(provisional, virtualization) && (2)
\end{aligned}
$$

where $x_{se,n}$ is 1 if the service *se* is assigned to device *n*; 0 otherwise.

Firstly, the NDF (see Eq. (2)) maximizes the number of supported devices (*s*), according to a series of restrictions: (1) *s* contains the application services already supported by the infrastructure; and (2–3) the devices have RAM and disk enough to allocate the supposedly supported services. Secondly, the result of this maximizing problem is post-processed, identifying the non-supported services (4) and creating a set of provisional devices from the non-supported characteristics (5). Finally, the NDF operates with the devices contained in *provisional*, and returns the minimal set of devices with the configuration the infrastructure needs to support the application, adapting its characteristics according to the boolean value *virtualization*, given as an input.

### 3.3. Module 3: Edge-Deployment Alternatives Finder (EDAF)

Once the application and infrastructure are configured, the aim of the EDAF is to inform developers about the different deployment options available that guarantee the fulfillment of the application requirements. In practice, infrastructures are commonly shared among many applications; some devices of the infrastructure may be crucial for some applications (e.g., due to their peripherals), while others applications only demand the use of a certain computational power of them. For this reason, the EDAF module provides a mechanism to avoid, if possible, the use of some devices that are indispensable for others applications.

To control the hardware and software requirements of services increasing the opportunities and alternatives of deployment, the application services are split into tasks. Thus, the EDAF module operates at task level (Wang et al., 2019), being the most suitable granularity level to provide flexible solutions of deployment (Mao et al., 2017). The order of execution and data inter-dependencies among tasks are represented using task-call graphs (Mao et al., 2017) (see the input of the EDAF in Fig. 1). Nodes of the graph represent the tasks, while the edges represent relationships between tasks (by means of data produced and required). Specifically, each node the graph includes the hardware and software requisites of the task it represents, including the required CPU cycles, and the minimum RAM and required. On the other hand, the graph edges are labeled with the amount of data transmitted between tasks.

The characteristics of the nodes modeled in Section 4, along with the ones of the tasks, are used to predict the latency associated for computation and data transmission (Dinh et al., 2017; Zhang et al., 2016; Mao et al., 2017). The first expression of Eq. (3) shows the expression used to calculate the computation time (s) of a task *i* by node *n* ($tComp_{i,n}$), given by the relationship between the number of CPU cycles associated to the task *i* ($w_i$)–which value is estimated (Melendez and McGarry, 2017)–and the CPU power of node *n* (cycles per second, $F_n$). The communication time (in seconds) of the output data that task *i* sends to task *j* is given by the sum of the relationship between the amount of bits to send ($c_{i,j}$) and the minimum between the upload transmission rate ($R_n^{Tx}$) of the sender node ($n$) and the download transmission rate ($R_z^{Rx}$) of the receiver node ($z$) for a given connection type ($ct$)–e.g., WiFi 2.4 GHz, plus the propagation delay (s) between *n* and *z* ($t_{n,z}^{prop}$)–assumed as 0 between edge devices (Mao et al., 2017; Dinh et al., 2017)–as seen in the second expression of Eq. (3):

$$
\begin{aligned}
&tCompu_{i,n} = x_{i,n} \frac{w_i}{F_n}\\
&tCommu_{i,j,n,z,ct} = x_{i,n} h_{i,j} \left( \frac{c_{i,j}}{Min(R_{n,ct}^{Tx}, R_{z,ct}^{Rx})} + t_{n,z}^{prop} \right) \qquad (3)
\end{aligned}
$$

where $x_{i,n}$ is 1 if *task i* is assigned to *node n*, 0 otherwise; $h_{i,j}$ is 1 if *node n* and *z* are not the same. The propagation delay is set as the half of the mean round trip time (RTT) obtained by pinging from *n* to *z*, and considered like constant (Dinh et al., 2017).

While the NDF evaluates the time restrictions at service level, it makes no sense do the same with the tasks, as application's services are formed by groups of tasks. Instead, the EDAF groups the tasks responsible for providing each service and give them a maximal amount of time to be completed (i.e., a time restriction). Tasks with time restrictions are collected in lists named *tr*; this data structure not only refers the tasks involved in each of them, but also contains the maximal time to be completed ($tr_{time}$). *Tr* contains all the groups of tasks with time restrictions.

In addition to the expressions shown in Eq. (3) and the function *meets* previously presented, the EDAF uses:

- *nUsers (assignment, nodes):* returns the maximum number of users supported by the infrastructure according to the tasks assignment and nodes capabilities.
- *connType (i, j):* returns the connection type assigned to communicate tasks *i* and *j*.

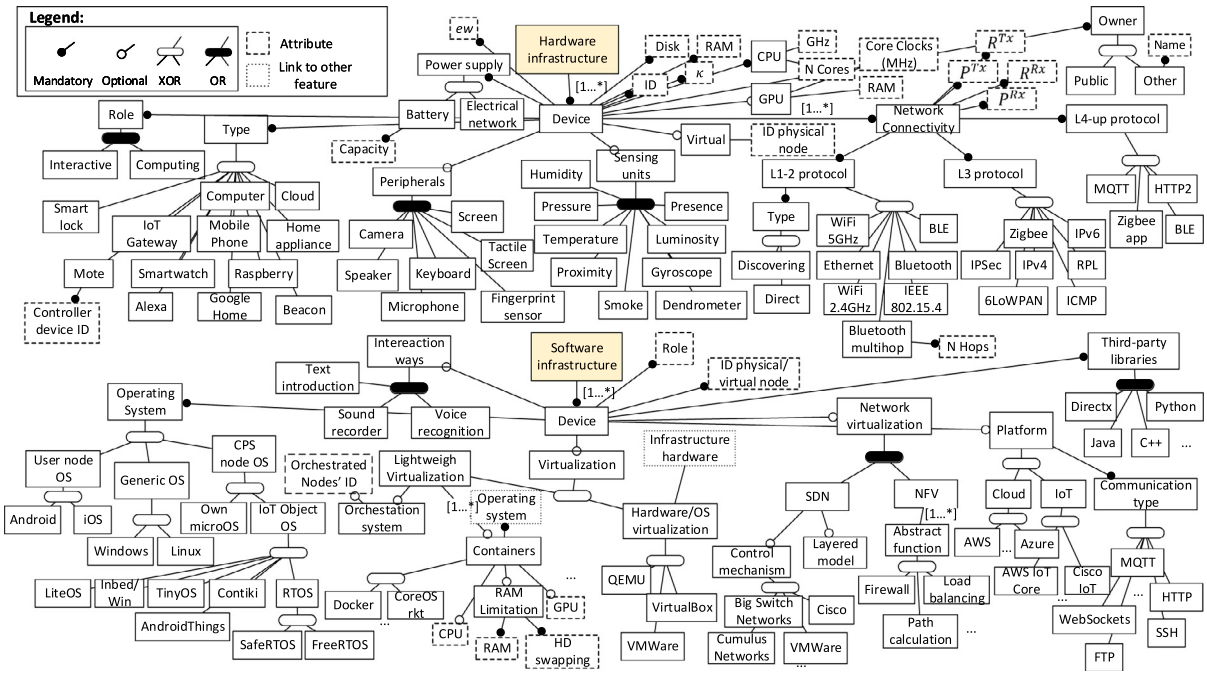


**Fig. 2.** Feature models of the hardware and software information of the infrastructure's devices.

As an input, the EDAF receives the application's task-call graph (representing $\tau$ the set of tasks and $c_{i,j}$ the amount of data to transmit between tasks $i$ and $j$); the time restrictions between tasks ($Tr$); the configured infrastructure FMs (hardware and software)–modeled in the set $N$; the devices to avoid (if any, $nToAvoid$); and, optionally, the minimal number of users to support ($nUsers$)–considering RAM and usage. This module returns the different options of tasks assignment, including resources allocation, communication type used in each data transmission, and the maximal number of users supported. If the current infrastructure does not support the application (there is not a solution), the EDAF returns an empty set. Once the tasks are assigned to the devices, their resources can be limited in practice using virtualization, containers (e.g., Docker), or unikernel based solutions (Bratterud et al., 2015).

$$
\begin{aligned}
\textbf{Subject to}: \quad & N = N \setminus nToAvoid && (1)\\
& \forall i \epsilon \tau : \sum_{n \epsilon N} x_{i,n} = 1 && (2)\\
& \forall n \epsilon N : \sum_{i \epsilon \tau} x_{i,n} i_{RAM} \leq n_{RAM} && (3)\\
& \forall n \epsilon N : \sum_{i \epsilon \tau} x_{i,n} i_{disk} \leq n_{disk} && (4)\\
& \forall n \epsilon N, i \epsilon \tau : x_{i,n} RAM_{i,n} nUsers \geq i_{RAM} x_{i,n} && (5)\\
& \forall n \epsilon N, i \epsilon \tau : x_{i,n} disk_{i,n} nUsers \geq i_{disk} x_{i,n} && (6)\\
& \forall n \epsilon N, i \epsilon \tau : x_{i,n} \wedge meets(n,i) \vee \neg x_{i,n} && (7)\\
& \forall (n,z) \epsilon N, (i,j) \epsilon \tau : x_{i,n} x_{j,z} \wedge c_{i,j} > 0 \\
& \quad \rightarrow connType(i,j) \in n_{con} \cap z_{con} && (8)\\
& \forall tr \epsilon Tr, (n,z) \epsilon N, i \epsilon \tau : x_{i,n} tCompu(i,n) + \\
& \quad \sum_{(j) \epsilon tr} h_{i,j} x_{j,z} tCommu(c_{i,j}, n, z, \\
& \quad connType(i,j)) \leq time_{tr} && (9)\\
\textbf{Return}: \quad & \forall i \epsilon \tau, n \epsilon N : x_{i,n} \rightarrow RAM_{i,n}; x_{i,n} \rightarrow disk_{i,n} \\
& \forall (i,j) \epsilon \tau : c_{i,j} \rightarrow connType(i,j) \\
& nUsers(x, N)
\end{aligned}
\qquad (4)
$$

The EDAF, formalized in Eq. (4), does the following: firstly, excludes the nodes to avoid from the set $N$ (1); (2) checks that every node have a node assigned for its execution; (3–4) assure that nodes allocate RAM and enough to execute their assigned tasks, while (5–6) verify that nodes have RAM and enough to serve $nUsers$; (7) checks that nodes accomplish the tasks' requirements, while (8) assures that nodes that have to send/receive data between them are using the same communication capability; (9) verifies the accomplishment of the time restrictions, considering the execution and communication times; finally, the EDAF returns the solution (tasks allocation, communication capabilities used and number of users supported). Unlike the rest of modules, the EDAF does not minimize or maximize any variable, but is executed multiple times, returning every existing alternative of deployment that meets the aforementioned conditions.

### 3.4. Module 4: Energy Consumption and Latency Minimizer (ECLAM)

The objective of the ECLAM is to provide the optimal subset of devices and tasks allocation to deploy the application that minimizes the energy consumption and/or execution latency, accomplishing the application requirements on the way.

The functions of latency and energy consumption to minimize are shown in Eq. (3) Eq. (5) respectively. Eq. (3) shows how is calculated the time required by computation and data transmission of a deployment solution. The energy consumption in the nodes is influenced by several factors, such as the usage of CPU (Central Processing Unit), storage, and RAM, being the CPU usage the most influential factor (Tran and Pompili, 2019) and the one in which most of EC approaches typically base their energy consumption models (Mao et al., 2017). The first expression of Eq. (5) shows the expression used in this work to predict the energy consumption of computation (J) required by node $n$ to compute task $i$ ($eCompu_{i,n}$). The energy consumed by task $i$ to communicate with task $j$ (using connection type $ct$, that will be WLAN or the Internet) is presented in second and third expressions of Eq. (5), that calculate the energy consumption for data sending and receiving in nodes $n$ and $z$ (sender and receiver nodes, respectively) (Mao et al., 2017):

$$eCompu_{i,n} = x_{i,n} \kappa_n w_i F_n^2 e w_n$$

$$eSend_{c_{i,j},n,ct} = x_{i,n} h_{i,j} P^{Tx}_{n,ct} \frac{c_{i,j}}{R^{Tx}_{n,ct}} ew_n$$

$$eRecp_{c_{i,j},z,ct} = x_{j,z} h_{i,j} P^{Rx}_{z,ct} \frac{c_{i,j}}{R^{Rx}_{z,ct}} ew_z \quad (5)$$

In contrast to other task-offloading approaches that typically focus on the energy consumption in the user device regardless the rest of the infrastructure (Chen et al., 2016; Chen and Yixue, 2018) , our proposal allows minimizing the global energy consumption; i.e., the total energy consumption of the infrastructure. At the same time, it allows to select the importance of saving energy in each device separately. Thus, if your intention is to minimize only the energy consumption in the user device, simply set the rest of *ew* (energy weight, see Expression (5)) to 0. This allows providing flexibility to the deployment solutions and ease to define policies according to developer's necessities.

The ECLAM module receives the same input as the EDAF, and the objective(s) sought (energy and/or latency). As output it returns the optimal tasks distribution among the infrastructure. In case of not being supported the application by the infrastructure, this module returns an empty set.

**Minimize** :
$$energy \rightarrow \forall(n,z)\epsilon N, i\epsilon\tau : x_{i,n} eCompu(i,n) + \sum_{j\epsilon\tau} \left( h_{i,j} eSend(c_{i,j}, n, connType(i,j)) + x_{j,z} eRecp(c_{i,j}, z, connType(i,j)) \right)$$
$$latency \rightarrow \forall(n,z)\epsilon N : \forall(i)\epsilon\tau : x_{i,n} tCompu(i,n) + \sum_{(j)\epsilon\tau} h_{i,j} x_{j,z} tCommu(i,j,n,z,connType(i,j))$$

**Subject to** :
$$N = N \setminus nToAvoid \quad (1)$$
$$\forall i\epsilon\tau : \sum_{n\epsilon N} x_{i,n} = 1 \quad (2)$$
$$\forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} i_{RAM} \leq n_{RAM} \quad (3)$$
$$\forall n\epsilon N : \sum_{i\epsilon\tau} x_{i,n} i_{disk} \leq n_{disk} \quad (4)$$
$$\forall n\epsilon N, i\epsilon\tau : x_{i,n} RAM_{i,n} nUsers \geq i_{RAM} x_{i,n} \quad (5)$$
$$\forall n\epsilon N, i\epsilon\tau : x_{i,n} disk_{i,n} nUsers \geq i_{disk} x_{i,n} \quad (6)$$
$$\forall n\epsilon N, i\epsilon\tau : x_{i,n} \wedge meets(n,i) \vee \neg x_{i,n} \quad (7)$$
$$\forall(n,z)\epsilon N, (i,j)\epsilon\tau : x_{i,n} x_{j,z} \wedge c_{i,j} > 0 \rightarrow connType(i,j) \in n_{con} \cap z_{con} \quad (8)$$
$$\forall tr\epsilon Tr, (n,z)\epsilon N, i\epsilon\tau : x_{i,n} tCompu(i,n) + \sum_{(j)\epsilon tr} h_{i,j} x_{j,z} tCommu(c_{i,j}, n, z, connType(i,j)) \leq time_{tr} \quad (9)$$

**Return** :
$$\forall i\epsilon\tau, n\epsilon N : x_{i,n} \rightarrow RAM_{i,n}; x_{i,n} \rightarrow disk_{i,n}$$
$$\forall(i,j)\epsilon\tau : c_{i,j} \rightarrow connType(i,j)$$
$$nUsers(x,N) \quad (6)$$

Eq. (6) formalizes the ECLAM, that minimizes the energy consumption and/or latency, subject to the same restrictions (1–9) that the EDAF (see Section 3.3). Nevertheless, this time the module only returns one solution. In case of receiving both objectives as an input (energy and latency), the ECLAM returns a trade-off between the both (De Moura and Bjørner, 2008).

We present a more detailed description of the output of the last two modules in Section 5. As the output provides the information needed to deploy the application, in case of using any Infrastructure as Code (IaC) tool, their output can be processed to transform it into a understandable input for the IaC tool used.

## 4. Extensible and reusable FMS for edge-based infrastructures

To deal with the high variability presented in applications and deployment infrastructures, we use multi-layer FMs (Rosenmüller et al., 2011). We propose the use FMs with cardinality (Czarnecki et al., 2005) and numerical attributes (Benavides et al., 2005) (see 1). CMCs, which are formed by logic formulas, maintain the coherence between the FMs of different layers. CMCs allow to comply with the application requirements as determine what hardware and software demands the application's services configured by the FMs from the infrastructure. Note that all these models are defined only once by the domain engineer, and can be reused to deploy different applications of the same family or domain.

As aforementioned, we use three FMs: one to model the application and two for the infrastructure. While the application FM will depend on the specific application, the FMs of the infrastructure can be extended and reused to be applicable to any scenario. In this section, we present two extensible and reusable FMs for infrastructures formed by cyber–physical systems. We define the hardware and software layers, as well as their CMCs.

### 4.1. FM of the hardware infrastructure

Devices are characterized by their type, network connections and, optionally, peripherals and sensing units associated. Computers, IoT Gateways, home appliances, smartphones, sensing motes and virtual assistants, such as Alexa, are some examples of the type devices that can be configured (see top of Fig. 2). Devices may play two roles inside the infrastructure which determine their function: *computing nodes* (capable to receive tasks offloaded from other nodes) and *interactive nodes* (used directly by the users). For instance, the role of a Google home device can be to interact with the user and/or receive offloaded tasks. Peripherals and sensing units will determine the hardware possibilities of the devices to interact with the environment.

The computational power of the devices is modeled by the CPU frequency (GHz) and the number of cores; the same with the GPU (Graphic Processing Unit). Memory capacities can be configured by the attributes RAM and disk. The communication capabilities are modeled by the typical networking layers or protocol stacks. Each network connection includes the uploading and downloading transmission rates ($R^{Tx}$ and $R^{Rx}$ respectively). All these features and attributes provide information enough to predict the latency of the application execution. Regarding the energy consumption of task computing and data transmission, the model contains several features to predict it. Concretely, $\kappa$ is a constant that depends on the hardware and directly influences the energy consumption of computing tasks (Zhang et al., 2013); other influencing factors are the CPU frequency and the time required for computation (Dinh et al., 2017; Zhang et al., 2016; Mao et al., 2017). Concerning the energy consumed by communication, factors such as $P^{Tx}$ and $P^{Rx}$ (W), which are the upload/download transmission powers (assumed like constants and calculated using power control systems Dinh et al., 2017), and the time required for transmission are used to predict it (Mao et al., 2017). Our model contains the attribute *ew* (energy weight), which value ranges between 0 and 1, used in our modules (Section 3.4) to determine how important is to save energy in the device (Cañete et al., 2019).

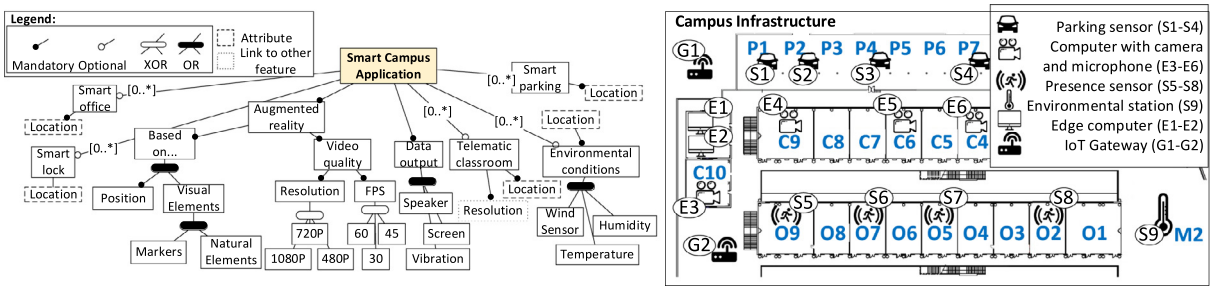


**Fig. 3.** Feature model of the smart campus application (left). Disposal and type of the devices of the infrastructure (right).

### 4.2. FM of the software infrastructure

This layer models software components like the operating system, third party libraries, and also the interaction modes, which models the data input/output channels (e.g., text introduction, voice recognition, etc.). In addition, software virtualization (i.e., virtualization and containment technologies supported), which extends software device capabilities, is also included, distinguishing between hardware/OS virtualization and lightweight virtualization based on containers (Plauth et al., 2017). In the first case, software applications run on virtualized hardware, which is considered as a device. The same physical device can include several virtual machines (VMs), that are constrained by the hardware resources of the device—modeled by linking these features with the hardware feature model (Spinczyk and Beuche, 2004). This configuration allows to select the technology used for virtualization with the aim of deploying VMs on demand. On the other hand, containers emulate the services of an operating system, allowing to define runtime usage limitations related to memory, CPUs, and GPUs (although is not mandatory). Once again, this information allows to launch new containers on demand. Although containers technologies (e.g., Docker) allow workload balancing among containers instantiated in the same device, they do not allow it among containers running on different devices by themselves. To do so, it is necessary to defined orchestration systems by identifying and configuring the devices subscribed to the orchestration (e.g., Kubernetes) (Casalicchio, 2019). Regarding the network connectivity, devices may have one or more virtual networks associated. Cloud devices are described by the software platform and communication type, which can be located both in the core and in the edge of Internet, differing in latency and computation features.

### 4.3. Constraints between hardware and software layers

Hardware and software characteristics are highly dependent, being the software conditioned by the hardware. Thus, it is necessary to define the CMCs between the feature models previously presented.

The user interaction (presented in the software layer) is restricted by the peripherals of the hardware configuration. Despite that, we consider the presence of some hardware characteristics insufficient to assure an interaction method (e.g,. acoustic sensing motes do not allow voice recognition). Thus, we consider several restrictions that our models must contain:

```
play sounds requires speaker                                    1
text introduction requires keyboard or tactile screen           2
voice recognition or sound recorder requires microphone         3
computing node or interactive node requires operating system    4
cloud requires platform                                         5
```

Regarding virtualization, technologies based on VMs emulate not only the software, but also the hardware of the devices. Nevertheless, most of the hardware features are inherent to the device and cannot be modified in the VM configuration (such as the power supply, type of device or the constant of power consumption ($\kappa$)). On the other hand, some of them are limited by the hardware, like the CPU, GPU, RAM and disk. Finally, several features may be restricted in terms of accessibility by the VM (e.g., sensing units, peripherals, connectivity, etc.). All these restrictions are collected by the constraint rules presented below:

$$\left(\sum_{i=1}^{device.VMs} device.VM_i.RAM\right) \leq device.RAM \quad (1)$$

$$\left(\sum_{i=1}^{device.VMs} device.VM_i.GPU_{RAM}\right) \leq device.GPU_{RAM} \quad (2)$$

$$\left(\sum_{i=1}^{device.VMs} device.VM_i.disk\right) \leq device.disk \quad (3)$$

$$\left(\sum_{i=1}^{device.VMs} device.VM_i.CPU_{Ncores}\right) \leq device.CPU_{Ncores} \quad (4)$$

$$\forall\ VM \in device: \quad (5)$$

$$VM.peripherals \subseteq device.peripherals \ \texttt{and} \quad (6)$$

$$VM.sensingUnits \subseteq device.sensingUnits \ \texttt{and} \quad (7)$$

$$VM.CPU \leq device.CPU \ \texttt{and} \quad (8)$$

$$VM.GPU_{clocks} \leq device.GPU_{clocks} \quad (9)$$

$$\texttt{and VM.networkConnectivities} \quad (10)$$

$$\subseteq \texttt{device.networkConnectivities and} \quad (11)$$

$$\texttt{VM.powerSupply = device.powerSupply} \quad (12)$$

$$\texttt{and VM.type = device.type} \quad (13)$$

where the first four assure that device has enough resources – in terms of CPU cores, GPU memory, disk, and RAM capacity – to allocate all the VMs configured on it. The rest of the constrains guarantee the correlation between physical and virtualized devices concerning peripherals, sensing units, CPU and GPU frequencies, network connectivity and power supply.

## 5. Proof of concept

This section demonstrates the feasibility of our approach, applying it to a real IoT scenario and evaluating its scalability.

### 5.1. Illustrative example

Consider an academic campus where several devices, those typical of CPSs, are geographically distributed serving different applications. The campus infrastructure includes sensing units, IoT gateways, computers, cloudlets, and dedicated cloud servers, scattered across the campus. These devices are not using all their computation and communication capacities (or even are suspended most of the time). All of them are located at the edge of Internet, connected to the campus institutional access network. Their use is shared among several deployed applications, which are managed by different research groups and academic departments.

Fig. 3 shows the geographical location of the devices on a map of the campus (on the right side). Concretely, the map includes the location of two Meshlium IoT-Gateways[1] (G1 and G2), four motes with parking sensors (S1–S4), four motes with presence sensors (S5–S8), one mote with temperature, humidity and wind sensing units (S9), two computers without special peripherals or

[1] http://www.libelium.com/development/meshlium/documentation/.

sensing units associated (E1 and E2), and four computers with camera and microphone (E3–E6).

A new version of a smart application for mobile devices (app for short) to guide students, staff and visitors across the college campus is going to be released. This app supports augmented reality functionality to enrich user experience through the information obtained by sensing units, providing environmental information or notifying the presence of professors in their offices in real time. In addition, this new version allows attending video conferences and online classes. The FM of the application family is shown in the left part of Fig. 3.

Firstly, the developer is quite concerned about the application's functionalities that the infrastructure supports (see Fig. 3). For this reason, executes the **Application Variability Adaptor (AVA)**,[2] obtaining as a result:

```
Non-Supported: {Smart lock}                                          1
Feature: Smart parking, Location: {P1, P2, P4, P7}                   2
Feature: Smart office, Location: {O2, O5, O7, O9}                    3
Feature: Environmental information, Location: {M2}                   4
      Restrictions: {Owner: TST Group}
Feature: Telematic Classroom, Location: {C4,C6}                      5
      Restrictions: {Video Quality <= 720p}
Feature: Telematic Classroom, Location: {C9, C10},                   6
      Restrictions: {Video Quality <= 480p}
```

Now, the developer knows that the current infrastructure does not support any smart lock. Nevertheless, the infrastructure is capable of monitoring three parking places (P1, P2 and P4), four smart offices (O2, O5, O7 and O9) and the context information through the environmental station located at M2 (owned by the TST research Group). In addition, the smart campus application would support video streaming classes C4, C6, C9 and C10, being the maximal video quality of 720p for classes C4 and C6, and of 480p for C9 and C10.

Now, suppose that the app must allow the teacher to facilitate students the remote assistance to classroom C8 by creating a virtual conference room and transmit audio and video streaming in 1080p resolution, opening the door C8 to the face-to-face attendants using a NFC lock, and monitoring the state of the parking P6. The developer does not know the minimal set of new devices needed to run the desired application on the infrastructure. The answer entail the evolution of the infrastructure by purchasing the devices given as an output of our second module, the **New Devices Finder (NDF)**:

```
Node 1: {Type: Mote, Sensing units: Parking, Location: P6}      1
Node 2: {Type: Smart lock, Location: C8}                        2
Node 3: {Type: Computing, Peripherals: {Camera, Microphone},    3
Location: C8, Restrictions: {Transmission rate >= 3686400}}     4
```

Now, the developer can inform the infrastructure administrator that the infrastructure requires three new devices to support the desired functionalities of the app. Concretely, a parking sensor located at P6 (Node 1); a smart lock, located at C8 (Node 2); finally, a computing node located at C8, with camera and microphone as peripherals, and a transmission rate greater than or equal to 3686400 bits/s (Node 3).

Thanks to a new research project budget, an investment is done to upgrade and improve the college infrastructure. The new devices required by new research project (highlighted by the NDF) are included in the infrastructure, while the resources offered by the existing ones can improve their capabilities. The developer plans to deploy the new application on the infrastructure, but does not know how to distribute the application tasks to accomplish the QoS.

[2] Source code available at: https://doi.org/10.5281/zenodo.4744293.

The answer is provided by the use of the third module **Edge-Deployment Alternatives Finder (EDAF)**. Concretely, the functionality of this application (configured from the FM shown in the left side of Fig. 3) is decomposed in 21 different tasks (t): t1–t3, responsible for monitoring three parking places (P1, P2 and P4); providing video sharing in classrooms C4, C9 and C10 (t4–t6); notifying the presence of professors in offices O2, O5 and O7 (t7–t9); and collecting environmental conditions (t10) from the environmental station located at M2, which provides updated information about the weather. Apart from these functionalities, the augmented reality component enriches the real world according to the user's location, which is determined using SLAM (Simultaneous Location and Mapping) technology (Sarker et al., 2019), provided by t11–t21. These kind of applications are specially attractive to apply edge computing, as the tasks are computational-intensive and delay-sensitive and their execution on mobile devices is generally prohibitive in terms of battery cost when satisfying users' expectations in terms of latency (Sarker et al., 2019; Al-Shuwaili and Simeone, 2017). Concretely, 11 of these 21 tasks correspond to activities for enriching the reality, 3 of which must be executed in the user node—the tasks responsible for camera calibration, capture frames and overlay context (t11, t13 and t17). Regarding the available resources of the infrastructure (formed by the 17 devices depicted in Fig. 3 and the user's smartphone), their sensing units, peripherals, location and communication capabilities were previously described in this section; their CPU frequency, transmission rates and other characteristics have been randomly generated. In that case, nodes reserve 3 Gb of RAM and 10 Gb of disk to execute offloaded tasks.

An output example of the EDAF for the case presented on Fig. 3 is:

```
SOLUTION 1. Users supported: 14 {                                    1
 User node -> Tasks:{t11,t13,t14,t17,t19}, RAM:195;                  2
 S1 -> Tasks:{t1}; S2 -> Tasks:{t2}; S3 -> Tasks:{t3};               3
 E6 -> Tasks:{t4}; E4 -> Tasks:{t5}; E3 -> Tasks:{t6};               4
 S8 -> Tasks:{t7}; S7 -> Tasks:{t8}; S6 -> Tasks:{t9};               5
 S9 -> Tasks:{t10}; G1 -> Tasks:{t15,t16,t18,t20,t21},               6
 RAM:210, Disk:395; G2 -> Tasks:{t12}, RAM:5, Disk:20;}              7

SOLUTION 2. Users supported: 60 {                                    8
 User node -> Tasks:{t11,t12,t13,t15,t16,t17,t18,t21}, RAM:          9
 280; S1 -> Tasks:{t1}; S2 -> Tasks:{t2}; S3 -> Tasks:{t3};          10
 E6 -> Tasks:{t4}; E4 -> Tasks:{t5}; E3 -> Tasks:{t6};               11
 S8 -> Tasks:{t7}; S7 -> Tasks:{t8}; S6 -> Tasks:{t9};               12
 S9 -> Tasks:{t10}; G1 -> Tasks:{t19}, RAM:50, Disk:80;              13
 G2 -> Tasks:{t14}, RAM:30, Disk:50; E1 -> Tasks:{t20},              14
 RAM:50, Disk:80;}                                                   15
...                                                                  16
```

In Solution 1, the tasks responsible for monitoring the parking places (t1–t3) have been assigned to their corresponding motes (S1–S3). The same occurs with the tasks responsible for video sharing (t4–t6, assigned to E3–E6 depending on their locations), notifying the presence of professors (t7–t9, assigned to S6–S8), and collecting environmental information (t10, assigned to S9). The task responsible for reality augmentation is distributed among the user node, G1 and G2: the node of the user executes tasks t11, t13, t14, t17 and t19, using 195 Mb of RAM; G1 allocates t15, t16, t18, t20 and t21, reserving 210 Mb of RAM and 395 Mb of disk; finally, G2 executes t12 and reserves 5 Mb of RAM and 20 Mb of disk. Note that the user node has the entire application installed, delegating tasks according to the infrastructure characteristics and adapting its execution accordingly (Cañete et al., 2020). Thus, it does not reserve disk storage. The number of users supported by this solution is 14 (limited by G1's workload).

In the second solution, tasks that require a specific location and nodes (t1–t10) have the same devices assigned as in Solution 1. Nevertheless, this time the distribution of the augmented reality tasks differs from the first solution: the user node has t11, t12, t13, t15, t16, t17, t18 and t21 assigned, requiring 280 Mb of RAM; G1 executes t19, reserving 50 Mb of RAM and 80 Mb of

disk; t14 is assigned to G2, that allocates 30 Mb of RAM and 50 of disk; finally, t20 is assigned to E1, which allocates 50 Mb of RAM and 80 Mb of disk. Considering that nodes reserves 3 Gb of RAM and 10 Gb of disk to execute offloaded tasks, this time the deployment solution is able to attend the offloading requirements of 60 users—being G1 the node that requires more resources once again, apart from the user node.

A few months after the application deployment, the University starts a new plan to reduce its carbon footprint, demanding the reduction of energy consumption in their facilities. Thus, the developer executes the **Energy Consumption and Latency Minimizer (ECLAM)**, with the energy consumption as optimizer function. Once again, the peripherals, location and sensing units of the devices are shown in Fig. 3, while their CPU power, RAM, disk and transmission characteristics have been randomly generated. The output of this module is:

```
Minimizing the energy consumption. Users supported:21 {                1
 User node -> Tasks:{t11,t13,t15,t16,t17,t19}, RAM:225;                 2
 S1 -> Tasks:{t1}; S2 -> Tasks:{t2}; S3 -> Tasks:{t3};                  3
 E6 -> Tasks:{t4}; E4 -> Tasks:{t5}; E3 -> Tasks:{t6};                  4
 S8 -> Tasks:{t7}; S7 -> Tasks:{t8}; S6 -> Tasks:{t9};                  5
 S9 -> Tasks:{t10}; G1 -> Tasks:{t14,t18,t20}, RAM:140,                 6
 Disk:210; G2 -> Tasks:{t12,t21}, RAM:45, Disk:90;}                     7
```

As in case of the EDAF, the ECLAM assigns the tasks with special requirements in terms of location (t1–t10) to the only nodes that achieve this condition in the infrastructure. Nevertheless, the deployment differs from the one provided by the EDAF in the rest of assignments: t11, t13, t15, t16, t17, t19 are assigned to the user node, reserving 225 Mb of RAM; t14, t18, t20 to G1, allocating 140 Mb of RAM and 210Mb of disk; finally, G2 allocates t12 and t21, consuming 45 Mb of RAM and 90 Mb of disk. Typically, IoT Gateways are well balanced in terms of computational power/energy consumption. That is the reason why the ECLAM assigns the majority of the delegable tasks to this kind of devices. The number of users supported by this deployment solution is 21.

After a few months, the developer checks the satisfaction surveys that the application includes, realizing that some users are quite concerned about the performance. Although the current deployment satisfies the requirements defined at configuration time, it is possible to increase the QoS changing the distribution of tasks. Hence, the developer executes the ECLAM newly, but this time optimizing both the energy consumption and latency of the application, obtaining as an output:

```
Trade-off energy consumption and latency. Users supported:16 {         1
 User node -> Tasks:{t11,t13,t15,t16,t17,t19}, RAM:225;                 2
 S1 -> Tasks:{t1}; S2 -> Tasks:{t2}; S3 -> Tasks:{t3};                  3
 E6 -> Tasks;{t4}; E4 -> Tasks:{t5}; E3 -> Tasks:{t6};                  4
 S8 -> Tasks:{t7}; S7 -> Tasks:{t8}; S6 -> Tasks:{t9};                  5
 S9 -> Tasks:{t10}; G1 -> Tasks:{t12}, RAM:5, Disk:20;                  6
 E1 -> Tasks:{t14,t18,t20,t21}, RAM:180, Disk:280}                      7
```

The assignment for tasks (t1–t10) has not changed, as well as the tasks assigned to the user node (t11, t13, t15, t16, t17, t19). Nevertheless, G1 has only t12 assigned, allocating 5 Mb of RAM and 20 Mb of disk, while G2 is not involved in the deployment. The majority of delegable tasks (t14, t18, t20, t21) have been assigned to E1, that reserves 180 Mb of RAM and 280 Mb of disk. Although not as energy efficient as the IoT gateways (as G1), edge computers (as E1) are usually more powerful, being this phenomenon reflected in the output of the ECLAM.

### 5.2. Reduction in the energy consumption and latency obtained by the ECLAM

The reduction in the energy consumption and execution time of the deployments returned by the ECLAM will highly depend on the application and infrastructure. In this section we compare the reduction in the execution time (s) and energy consumption (J) achieved by the ECLAM (for a concrete randomly infrastructure and application) with those obtained in a random assignment—that accomplishes the functional and non-functional application's requirements. With this purpose, we develop a *Benchmark*[3] version of the ECLAM, that allows setting the number of devices and tasks. We set the number of nodes in 30, whose characteristics have been arbitrary selected (on each experiment). These randomly generated characteristics involve the RAM, disk, CPU frequency, transmission capabilities, location, peripherals, sensing units, interaction ways, and operating system. Experiments consider infrastructures formed by 20, 30 and 40 nodes. To see to what extend the number of tasks may infer in the benefits obtained, we evaluate applications formed by 10 and 30 tasks in each case. Note that an application with fewer tasks may be more demanding that other with more tasks (resulting in more workload on the nodes). This phenomenon is considered in our experiments, since application characteristics are randomly generated. Experiments have been performed 30 times.

Table 1 shows the results, split in three rows: the first two for mono-objective optimizations (energy and time respectively), while the third optimizes both objectives at the same time. Results show that, when only the energy consumption is minimized (first row), the ECLAM achieves up to 51.12% of reduction in the energy consumption (considering all nodes). Table 1 shows that the more tasks the application has, the more alternatives of deployment we have ($nodes^{tasks}$) and, consequently, the more reduction in the energy consumption the ECLAM is able to obtain. When the objective is to minimize the latency (second row), the module obtains up to 28.37% of time reduction. Once again, the number of alternatives of deployments seems to infer in the reduction obtained. In case of multi-objective optimizations, the operation returns a trade-off between the both (De Moura and Bjørner, 2008). This time, the ECLAM continues providing good results, although not as good as in the case of single objective problems. In general terms, we conclude from our experiments that the more alternatives of deployment we have, the more benefit the ECLAM is able to obtain. We have also included a graphical view of these results on the left side of Fig. 4, where you can see at glance that the energy saving is greater than 40% in almost all cases. And that saving energy not only does not implies a penalty in the execution time when we include it as part of the optimization objective, but also it can be reduced at the same time as energy, although not so much.

### 5.3. Scalability

The time needed by our modules to provide a solution varies according to the size of the problem (Sundermann et al., 2020). This section evaluates their execution time for different problem sizes. With this purpose, we develop a *Benchmark* version of the modules, which allows setting the number of features and devices in case of Module 1 (AVA), services and devices for the second module (NDF), and tasks and devices in the case of Modules 3–4 (EDAF, ECLAM). Each experiment is performed 30 times on one thread of an AMD Ryzen 7 1700X processor.

In all the experiments the infrastructure considered is formed by 30 different devices with arbitrary characteristics (set on each experiment). As in Section 5.2, these random characteristics involve software and hardware components, as well as the location. Table 2 and Fig. 4 (right) show the results.

For the first module (AVA) the number of features has been incremented up to 100, being the 20% attributes and clonable features. Experiments show that the AVA module returns a result almost instantaneously in most cases, requiring around 4.2 s in the

[3] Source code available at: https://doi.org/10.5281/zenodo.4744293.

**Table 1**
Reduction in energy consumption and/or execution time obtained by the ECLAM.

| Problem size (D/T) | | | 20/10 | 20/30 | 30/10 | 30/30 | 40/10 | 40/30 |
|---|---|---|---|---|---|---|---|---|
| O: E[a] | Energy reduction | Mean (%) | 40.59 | 51.12 | 40.58 | 44.05 | 43.11 | 48.32 |
| | | Std | 15.26 | 7.31 | 13.57 | 27.82 | 26.74 | 15.02 |
| O: Ti | Time reduction | Mean (%) | 24.85 | 26.81 | 25.20 | 28.37 | 27.26 | 28.15 |
| | | Std | 10.02 | 8.05 | 3.10 | 9.83 | 7.29 | 5.89 |
| O: E + Ti | Energy reduction | Mean (%) | 41.11 | 38.52 | 35.57 | 39.38 | 40.38 | 42.12 |
| | | Std | 2.66 | 10.31 | 19.67 | 4.27 | 2.43 | 9.76 |
| | Time reduction | Mean (%) | 1.26 | 5.04 | 6.55 | 4.05 | 7.25 | 8.06 |
| | | Std | 0.71 | 2.02 | 3.47 | 1.72 | 4.52 | 3.03 |

[a]O: Objectives. E: Energy consumption. Ti: Execution time.

**Table 2**
Modules' execution times.

| Problem size: AVA: (D/F) NDF: (D/S) EDAF/ECLAM: (D/T)[a] | | 30/10 | 30/20 | 30/30 | 30/40 | 30/50 | 30/60 | 30/70 | 30/80 | 30/90 | 30/100 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AVA (Module 1) | Mean (s) | 0.07 | 0.14 | 0.24 | 0.41 | 0.60 | 0.88 | 1.39 | 1.99 | 2.95 | 4.23 |
| | Std | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 | 0.03 | 0.03 | 0.03 | 0.04 | 0.03 |
| NDF (Module 2) | Mean (s) | 0.12 | 0.50 | 1.09 | 1.97 | 3.03 | 4.49 | 6.03 | 7.94 | 10.11 | 12.21 |
| | Std | 0.00 | 0.01 | 0.01 | 0.02 | 0.03 | 0.07 | 0.06 | 0.06 | 0.16 | 0.30 |
| EDAF (Module 3) | Mean (s) | 0.55 | 3.13 | 6.28 | 13.81 | 38.14 | 73.32 | 87.93 | 124.11 | 250.63 | 240.99 |
| | Std | 0.16 | 1.13 | 1.96 | 4.21 | 7.90 | 5.93 | 26.17 | 31.16 | 44.33 | 74.63 |
| ECLAM (Module 4) | Mean (s) | 4.33 | 16.31 | 38.44 | 78.69 | 149.01 | 177.93 | 208.58 | 345.89 | 387.96 | 777.12 |
| | Std | 1.09 | 3.58 | 8.35 | 18.53 | 22.28 | 43.34 | 17.75 | 117.39 | 33.75 | 122.22 |

[a]D: Devices. F: Features. S: Services. T: Tasks.

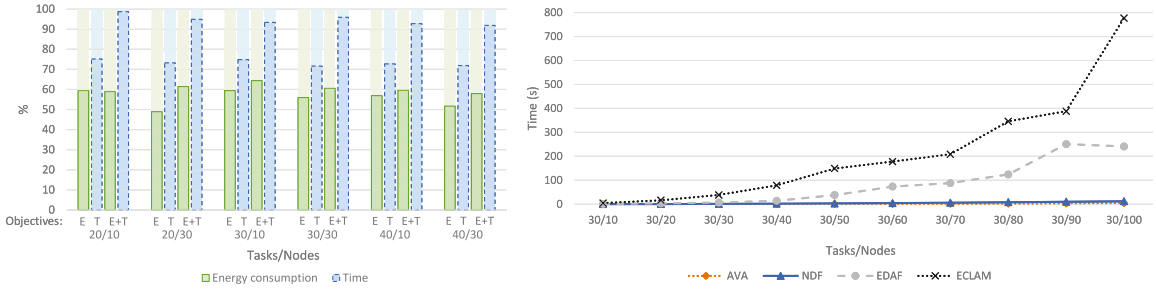


**Fig. 4.** Graphical representation of Tables 1 (left) and 2 (right).

worst case of a very big application formed by 100 heterogeneous features.

Regarding the experiments of the NDF module (second row), the number of services has been incremented from 10 to 100. Services characteristics as computational load, data to transmit, location requirements, type of device, peripherals, sensing units, operating system and interaction ways have been randomly set for each experiment. Results show that, for an application formed by 10 different services, the NDF needs around 0.12 s to return a solution, being 12.2 s in the worst case of an application formed by 100 different services.

In case of the EDAF, the number of tasks that form the application has been also incremented from 10 to 100. Apart from the characteristics randomly selected for services in the second module, the inter-dependencies among tasks and time restrictions between them have been also randomly generated. Third row of Table 2 collects the execution times required for this module to provide the first solution according to the problem size—number of tasks and nodes. The results show that the EDAF requires 0,55 s to provide a solution in case of an infrastructure of 30 different devices and an application composed of 10 tasks. As shown, the execution time increases with the number of tasks. For a very granulated application formed by 80 different tasks and an infrastructure of 30 heterogeneous devices (10th column of Table 2) this module takes around 124 s (i.e., 2 min and 4 s) to provide a solution. Nevertheless, we can conclude that the EDAF returns a solution in a moderate amount of time for typical IoT applications where the number of tasks is under 50 (less than 40 s).

Concerning the ECLAM (fourth row of Table 2), the experiment setup is the same as the EDAF's, while the objective(s) to optimize has been randomly set in each experiment (energy, latency or both). Results show that this module requires around 4 s to optimize a deployment for a scenario of 10 tasks and 30 heterogeneous devices. The execution time increments up to 208 s (i.e., 3 min and 28 s) for a very granulated application of 70 heterogeneous tasks (ninth column). During the experiments, we have realized that the ECLAM requires slightly more time to obtain solutions when both energy and latency objectives are considered—multiobjective. Note that this is the most complex module, as it solves a general assignment problem (GAP), which is NP-hard (Özbakir et al., 2010). Despite that, we conclude that it takes a reasonable amount of time to bring a solution.

## 6. Conclusions and future work

The application of Edge Computing imposes specific challenges on the developer to distribute the functionality of IoT and cyber–physical system meeting application requirements on the infrastructure to run properly. This work applies Multi Layer Feature Models to capture the variability of applications and the software and hardware infrastructure and support the deployment in edge-based environments of cyber–physical applications. With this multi-layered approach is possible to support the evolution of application and infrastructure independently. These models are used by four different modules implemented using a SMT solver. Application and infrastructure configuration is supported by the AVA and NFC modules. For a given and shared infrastructure, the AVA module assures that there will be enough resources for an application, informing developers about the feasible configuration alternatives. For a given application configuration, the NFC module calculates how to evolve the infrastructure with the minimal set of devices required to meet application requirements. Application deployment is supported by the EDAF and ECLAM modules. The EDAF is able to determine the alternatives of tasks distribution among the infrastructure nodes, while the module ECLAM calculates which is the task assignment that better pursue the reduction of the system energy footprint and/or latency. The utility of the four modules is showed by applying them to manage the evolution of an application and the infrastructure of a real IoT scenario. The execution time of our modules for different problem sizes (Sundermann et al., 2020) has been also evaluated using a Benchmark, and the conclusion is that our modules return a solution in a reasonable amount of time (in the order of minutes for the worst case).

For the ECLAM module the reduction in execution time and energy has been evaluated. Experiments show that ECLAM reduces by approximately 51% the energy consumption and up to 28% the execution time compared with the results obtained in a random assignment that accomplishes the functional and non-functional application's requirements.

As future work, we plan to integrate the ECLAM engine to Kubernetes, modifying its scheduler to select the deployment nodes according to the output of our optimizations (James and Schien, 2019).

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109 and by the projects co-financed by FEDER funds, Spain LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 (MCI/AEI) and RHEA P18-FR-1081. Funding for open access charge: Universidad de Málaga/CBUA.

## References

Abbas, A., Farah Siddiqui, I., Lee, S.U., Kashif Bashir, A., Ejaz, W., Qureshi, N.M.F., 2018. Multi-objective optimum solutions for IoT-based feature models of software product line. IEEE Access 6, 12228–12239.

Acher, M., Collet, P., Gaignard, A., Lahire, P., Montagnat, J., France, R.B., 2012. Composing multiple variability artifacts to assemble coherent workflows. Softw. Qual. J. 20 (3), 689–734.

Acher, M., Collet, P., Lahire, P., France, R.B., 2013. FAMILIAR: A domain-specific language for large scale management of feature models. Sci. Comput. Program. 78 (6), 657–681.

Ai, Y., Peng, M., Zhang, K., 2018. Edge computing technologies for Internet of Things: a primer. Digit. Commun. Netw. 4 (2), 77–86.

Al-Shuwaili, A., Simeone, O., 2017. Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. IEEE Wirel. Commun. Lett. 6 (3), 398–401.

Bagchi, S., Siddiqui, M.-B., Wood, P., Zhang, H., 2020. Dependability in edge computing. Commun. ACM 63 (1), 58–66.

Benavides, D., Trinidad, P., Ruiz-Cortés, A., 2005. Automated reasoning on feature models. In: Advanced Information Systems Engineering. Springer Berlin Heidelberg, pp. 491–503.

Bjørner, N., Phan, A.-D., Fleckenstein, L., 2015. νZ - AN optimizing SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 194–199.

Bratterud, A., Walla, A., Haugerud, H., Engelstad, P.E., Begnum, K., 2015. IncludeOS: A minimal, resource efficient unikernel for cloud services. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 250–257.

Bulej, L., Bures, T., Filandr, A., Hnetynka, P., Hnetynková, I., Pacovsky, J., Sandor, G., Gerostathopoulos, I., 2021. Managing latency in edge-cloud environment. J. Syst. Softw. 172, 110872.

Cañete, A., Amor, M., Fuentes, L., 2020. Energy-efficient deployment of IoT applications in edge-based infrastructures: A software product line approach. IEEE Internet Things J. 1.

Cañete, A., Amor, M., Fuentes, L., 2019. Optimal assignment of augmented reality tasks for edge-based variable infrastructures. In: 13th Int. Conf. on Ubiquitous Computing and Ambient Intelligence, UCAmI 2019, Toledo, Spain, December 2-5, 2019. In: MDPI Proceedings, vol. 31, MDPI, p. 28.

Cañete, A., Horcas, J.-M., Ayala, I., Fuentes, L., 2020. Energy efficient adaptation engines for android applications. Inf. Softw. Technol. 118, 106220.

Casalicchio, E., 2019. Container orchestration: A survey. In: Systems Modeling: Methodologies and Tools. Springer International Publishing, pp. 221–235.

Cecchinel, C., Mosser, S., Collet, P., 2016. Automated deployment of data collection policies over heterogeneous shared sensing infrastructures. In: 23rd Asia-Pacific Software Engineering Conference, APSEC 2016, Hamilton, New Zealand, December 6-9, 2016. IEEE Computer Society, pp. 329–336.

Chen, M., Dong, M., Liang, B., 2016. Joint offloading decision and resource allocation for mobile cloud with computing access point. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3516–3520.

Chen, M., Yixue, H., 2018. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. PP, 1.

Czarnecki, K., Helsen, S., Eisenecker, U., 2005. Formalizing cardinality-based feature models and their specialization. Softw. Process: Improv. Pract. 10 (1), 7–29.

De Moura, L., Bjørner, N., 2008. Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. In: TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg, pp. 337–340.

Dhungana, D., Grünbacher, P., Rabiser, R., Neumayer, T., 2010. Structuring the modeling space and supporting evolution in software product line engineering. J. Syst. Softw. 83 (7), 1108–1122, SPLC 2008.

Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.S., 2017. Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Trans. Commun. 65 (8), 3571–3584.

Elazhary, H., 2018. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. J. Netw. Comput. Appl. 128, 105–140.

Farahani, E., Habibi, J., 2019. Feature model configuration based on two-layer modelling in software product lines. Int. J. Electr. Comput. Eng. 9, 1–11.

Gámez, N., Fuentes, L., 2013. Architectural evolution of FamiWare using cardinality-based feature models. Inf. Softw. Technol. 55 (3), 563–580.

Geraldi, R.T., Reinehr, S., Malucelli, A., 2020. Software product line applied to the Internet of Things: A systematic literature review. Inf. Softw. Technol. 124, 106293.

Guo, J., White, J., Wang, G., Li, J., Wang, Y., 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. J. Syst. Softw. 84 (12), 2208–2221.

Holl, G., Grünbacher, P., Rabiser, R., 2012. A systematic review and an expert survey on capabilities supporting multi product lines. Inf. Softw. Technol. 54 (8), 828–852.

Huang, M., Liu, W., Wang, T., Liu, A., Zhang, S., 2020. A cloud–MEC collaborative task offloading scheme with service orchestration. IEEE Internet Things J. 7 (7), 5792–5805.

James, A., Schien, D., 2019. A low carbon kubernetes scheduler. In: Proceedings of the 6th International Conference on ICT for Sustainability, ICT4S 2019, Lappeenranta, Finland, June 10-14, 2019. In: CEUR Workshop Proceedings, vol. 2382, CEUR-WS.org.

Köksal, O., Tekinerdogan, B., 2019. Architecture design approach for IoT-based farm management information systems. Precis. Agric. 20 (5), 926–958.

Lettner, M., Rodas, J., Galindo, J.A., Benavides, D., 2019. Automated analysis of two-layered feature models with feature attributes. J. Comput. Lang. 51, 154–172.
Liu, F., Tang, G., Li, Y., Cai, Z., Zhang, X., Zhou, T., 2019. A survey on edge computing systems and tools. Proc. IEEE 107 (8), 1537–1562.
Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: The communication perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.
Melendez, S., McGarry, M.P., 2017. Computation offloading decisions for reducing completion time. In: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC). pp. 160–164.
Niewiadomski, A., Skaruz, J., Penczek, W., Szreter, M., Jarocki, M., 2014. SMT versus genetic and OpenOpt algorithms: Concrete planning in the PlanICS framework. Fund. Inform. 135, 451–466.
Özbakir, L., Baykasoğlu, A., Tapkan, P., 2010. Bees algorithm for generalized assignment problem. Appl. Math. Comput. 215 (11), 3782–3795.
Plauth, M., Feinbube, L., Polze, A., 2017. A Performance Survey of Lightweight Virtualization Techniques. pp. 34–48.
Pohl, K., Böckle, G., Linden, F., 2005. Software Product Line Engineering: Foundations, Principles, and Techniques.
Premsankar, G., Di Francesco, M., Taleb, T., 2018. Edge computing for the internet of things: A case study. IEEE Internet Things J. 5 (2), 1275–1284.
Rabiser, D., Prähofer, H., Grünbacher, P., Petruzelka, M., Eder, K., Angerer, F., Kromoser, M., Grimmer, A., 2016. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. Softw. Syst. Model. 17.
Reiser, M.-O., Weber, M., 2007. Multi-level feature trees. Requir. Eng. 12 (2), 57–75.
Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G., 2011. Multi-dimensional variability modeling. In: Fifth International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium, January 27-29, 2011. Proceedings. In: ACM International Conference Proceedings Series, ACM, pp. 11–20.
Sarker, V.K., Peña Queralta, J., Gia, T.N., Tenhunen, H., Westerlund, T., 2019. Offloading SLAM for indoor mobile robots with edge-fog-cloud computing. In: 2019 1st Int. Conf. on Advances in Science, Engineering and Robotics Technology (ICASERT). pp. 1–6.
Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50 (1), 30–39.
Shi, W., Pallis, G., Xu, Z., 2019. Edge computing [scanning the issue]. Proc. IEEE 107 (8), 1474–1481.
Spinczyk, O., Beuche, D., 2004. Modeling and building software product lines with eclipse. In: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications. In: OOPSLA '04, ACM, New York, NY, USA, pp. 18–19.
Sundermann, C., Thüm, T., Schaefer, I., 2020. Evaluating #SAT solvers on industrial feature models. In: Proceedings of the 14th Int. Conference on Variability Modelling of Software-Intensive Systems. In: VAMOS '20, ACM, New York, NY, USA.
Tran, T.X., Pompili, D., 2019. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans. Veh. Technol. 68 (1), 856–868.
Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z., Mohapatra, P., 2019. Edge cloud offloading algorithms: Issues, methods, and perspectives. ACM Comput. Surv. 52 (1), 2:1–2:23.
Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y., 2016. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. IEEE Access 4, 5896–5907.
Zhang, W., Wen, Y., Wu, D.O., 2013. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In: 2013 Proceedings IEEE INFOCOM. pp. 190–194.