This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

Energy-efficient Deployment of IoT Applications in Edge-based Infrastructures: A Software Product Line Approach

Angel Cañete, Mercedes Amor, and Lidia Fuentes

***Abstract*—In order to lower latency and reduce energy consumption, Edge Computing proposes offloading some computation intensive tasks usually performed in the Cloud onto nearby devices in the frontier/Edge of the access networks. However, current task offloading approaches are often quite simple. They neither consider the high diversity of hardware and software technologies present in edge network devices, nor take into account that some tasks may require some specific software and hardware infrastructure to be executed. This paper proposes a task offloading process that leans on Software Product Line technologies, which are a very good option to model the variability of software and hardware present in edge environments. Firstly, our approach automates the separation of application tasks, considering the data and operation needs and restrictions among them, and identifying the hardware and software resources required by each task. Secondly, our approach models and manages separately the infrastructure available for task offloading, as a set of nodes that provide certain hardware and software resources. This separation allows to reason about alternative offloading of tasks with different hardware and software resource requirements, in heterogeneous nodes and minimizing energy consumption. In addition, the offloading process considers alternative implementations of tasks to choose the one that best fits the hardware and software characteristics of available edge network infrastructure. The experimental results shows that our approach reduces the energy consumption in the user node by approximately 41%-62%, and the energy consumption of the devices involved in a task offloading solution by 34%–48%.**

***Index Terms*—edge computing; energy efficiency; Software Product Lines; Internet of Things.**

I. Introduction

THE popularity of the *Internet of Things* (IoT) [1] and *cyber-physical systems* [2] is non-stop growing. This kind of systems demands high performance computational resources capable of processing a large amount of data produced by a myriad of devices, ranging from smartphones and sensors to home appliances and all kind of wearables. To reduce the impact that the proliferation of IoT devices has in the global warming, international initiatives are promoting resource-efficient solutions. During the last years, Cloud computing has been the most popular infrastructure to provide a centralized solution capable of supporting the massive data storage, and with the high computational capabilities required to process IoT devices data [3]. The large capacity of cloud data centers has allowed this model to work properly during the last years, but paying a heavy cost in terms of energy and latency.

This led to the emergence of novel technologies and new paradigms based on Edge Computing (EC) [4][5], which shift the provision of computing services from the cloud, in the core of the Internet, to the edge of the Internet, proposing a more sustainable solution. The aim of the EC paradigm is to take advantage of the inactive computational cycles and unused storage space of edge devices placed at the Internet's frontier (e.g. routers or switches). With the goal of reducing the latency and resources demanded by the cloud, EC proposes to offload some computation intensive tasks from the cloud servers onto nearby edge devices located in a range of one or two hops. This means that part of the data processing and storage will be displaced to devices closer to where data and services are produced and consumed.

But taking advantage of EC is not an easy issue [6]. Firstly, IoT applications developers must deal with the decomposition of application functionality into a set of tasks and find out which of them could be allocated inside the Edge nodes [7][8] in an optimal way, and this is a challenging process. In existing EC approaches, the identification and separation of tasks are usually done manually, and are performed for a specific application [9][10][11][4]. However, it is noticeable that many of the tasks resultant from decomposing a IoT application are often common and recurrent.

After this, the developer has to match the tasks' requirements with the software and hardware features of available devices to decide how these tasks 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 task offloading solutions depending on their resource demand and delay sensitivity [9][8][7].

However, when making the task offloading decision, it should also be considered that some tasks may require some specific software and hardware infrastructure (e.g. a video camera). Thus, the task offloading constrained to the selection of a specific node with specific software and hardware resources is more complicated.

To address the challenges implied by the deployment of IoT applications in an heterogeneous edge-based infrastructure, we propose a task offloading process that focuses on answering the following two Research Questions (RQs):

A. Cañete, M. Amor, and L. Fuentes are with the Grupo CAOSD, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain. E-mail: {angelcv, pinilla, lff}@lcc.uma.es

- RQ1. How to model a variable set of (re)usable tasks resulting from the decomposition of IoT applications, including the definition of resources demand and dependencies on the hardware and software of Edge-based infrastructures.
- RQ2. How to automate the optimization of task offloading in a heterogeneous edge-based infrastructure to reach a certain Quality of Service (QoS) (e.g., minimize power consumption).

To address both research questions applying a Software Product Line (SPL) [12] approach would make a lot of sense, since it has been successfully applied to different domains to explicitly model variability. The variability model is the central artifact for SPLs, as it allows to specify the commonalities and variabilities shared by a set of products. The Feature Model (FM) [12] is the most widely used variability model and defines variability in terms of features and tree and cross-tree constraints. Considering that IoT applications and infrastructures share a large set of common functionalities, we use FMs to represent the decomposition of an application's functionalities into services and tasks, including the amount of resources required by them at runtime, addressing RQ1. Also, two FMs are defined to model the variability of hardware and software characteristics of the deployment infrastructure devices (e.g. mobile phone, WiFi, Android). The constraints of these models will be used to express relationships among features, just as a mobile phone can use WiFi and/or Bluetooth as communication technologies and uses the Android operating system. Therefore, RQ1 is addressed defining FMs to specify families of IoT applications that can be deployed in a variable set of devices present in a edge-based infrastructure represented by others FMs.

Once the decomposition of the application functionality into tasks is done, developers have to face the allocation of the resulting tasks preserving certain QoS (e.g. minimum energy consumption while maintaining a good QoS). This addresses RQ2. Currently, there are different task offloading solutions that reduce energy consumption, even in edge-based environments [13][14][7]. These algorithms address task allocation as an optimization problem: they accept as input a list of tasks tagged with costs (such as energy consumption, latency or computation resources) and constraints and generate an allocation of tasks to devices following a task offloading scheme. However, these offloading algorithms make some simplifications of the problem, by considering a closed set of devices, being all them of the same type, and with a fix amount of resources and also a predefined configuration [15][16][17]. Therefore, the information about devices managed by these algorithms does not fit well with most of real Edge computing scenarios, which are characterized by a high diversity of devices with different software infrastructure configurations. The software infrastructure variability is derived from the existence of different operating systems and middleware, including recent lightweight virtualization technologies [18]. In addition, since most of edge devices are shared by several applications, it is not realistic to consider a fix amount of resources [14][15].

On the contrary, the task allocation algorithm we present in this paper, used to deploy and re-deploy application tasks, properly deals with the different types of variability of hardware and software infrastructure installed in IoT devices. Also, since the infrastructure of IoT systems, including edge devices, are shared by several applications, our algorithm tracks the successive offloading of tasks in order to update the amount of available resources in each IoT device (usually load balancing manages the distribution of tasks of a single application [13]). In addition, our deployment algorithm finds the most suitable task offloading, while meeting some constraints such as low energy consumption, and considers the task dependencies, such as for example, input/output data or resource needed.

The automation of the assignment of tasks to devices is supported by two modules that assist developers in (a) the selection of the most appropriate set of tasks to carry out each application's functionality according to the features of the deployment infrastructure; and (b) the assignment of tasks and resources among the deployment infrastructure according to the current status of the devices. These modules use constraint programming, concretely a SMT (Satisfiability Modulo Theories) solver to provide an optimal solution to the task offloading problem. We have evaluated our proposal by configuring a real scenario of an edge-based infrastructure installed at the University of Málaga and comparing the energy consumption of the application with and without taking advantage of edge computing through our approach. The results of the evaluation show that up to 62% of reduction in the energy consumption in the user device can be obtained and up to 48% when the consumption of all nodes are considered (contemplating edge devices too) for our case study. The execution time of our modules is also evaluated for different problem sizes, obtaining that our system is capable of returning a solution in a reasonable amount of time.

II. Related work

This section discusses the state of the art in existing task offloading approaches. The use of SPL for modelling infrastructure is introduced.

The task offloading approaches found in the literature differ in the number of users supported and the type of edge nodes considered for offloading. In addition, many 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) [9][8][7]. Considering only homogeneous nodes, as these approaches make, reduces significantly the complexity of the problem, since only the code or certain task offloading decision (i.e., execute a specific computational load locally or remotely) are considered. A more realistic but complex scenario involves a heterogeneous set of edge nodes (in hardware and with different resource type available), shared among multiple users. In this scenario, other approaches consider, at least, tasks allocation. Task allocation involves deciding not only whether to execute a computational load locally or remotely (code offloading decisions), but also which node is the most adequate to offload it. Consequently, task allocation comprises resource management, and involves modelling the nodes' resources and perform the task offloading to provide a solution.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

TABLE I
COMPARISON OF TASK OFFLOADING APPROACHES FOR HETEROGENEOUS DEVICES

Approach	Design objective(s)	Optimization focus	Techniques[a]	Task model	Granularity level	Devices' characteristics considered	Solution mechanism
[7]	Overall energy and latency	User, data flow	TA	Binary offloading	Task	Hardware	Heuristic algorithm
[19]	Energy and latency	User	TA, DVFS	Partial offloading	Service	Hardware	Relaxation-based algorithm
[20]	Successful offloading probability	User	TA	Partial offloading	Task	Hardware	Heuristic algorithm
[21]	Energy	User + Edge	TA	Partial offloading	Task	Hardware	Distributed algorithm admitting the Nash equilibrium
[22]	Latency	User	TA, CC	Binary offloading	Task	Hardware	Distributed matching algorithm
[23]	Cost	User + Edge	TA, CM	Binary offloading	Application	Hardware	Markov decision problem
[24]	Energy and latency	User	TA, CM	Binary offloading	Task	Hardware	Heuristic two-stage algorithm
Our approach	**Energy**	**Customized**	**TA, AA**	**Partial offloading**	**Task**	**Hardware and software**	**SMT based algorithms**

[a] TA: Tasks Allocation; DVFS: Dynamic Voltage and Frequency Scaling; CC: Computation Caching; CM: Computation Migration; AA: App Adaptation

Table I overviews a comparison of approaches for task offloading. Concretely, Table I compares these approaches in terms of: the objective or objectives that drive the task allocation (second column); the type of node(s) for which the optimization is intended (user, user and edge, or customized by the developer; third column); the edge computing techniques applied (fourth column); the task model used (fifth column), which is mainly characterized by a binary or partial offloading scheme [9]. While the binary offloading does not allow the partition of tasks, a partial offloading task model allows splitting tasks into simpler components, providing much more flexible solutions; the sixth column indicates the granularity level supported by each approach, i.e., the size of the computational load that is considered for offloading (tasks, services, or applications); the seventh column refers to the nodes characteristics that are considered by the tasks allocation algorithm; and, finally, the eighth column shows the method used to find a solution for the task allocation problem.

Table I details the characteristics of the following works: In [7] (first row), an IoT-mobile edge computing task offloading service orchestration scheme is proposed. The objective is to reduce the network transmission. A heuristic algorithm solves the differentiated cloud-edge binary offloading decisions, using an optimization function based on communication energy consumption, computation energy consumption, and task delay models. The evaluation shows that the total energy consumption of MEC servers is 20% smaller than the cloud server for big data-based applications. Dinh et al. [19] (second row) propose to set up the CPU frequency of the edge nodes according to the computational demands of the offloaded tasks, using Dynamic Voltage and Frequency Scaling techniques. Tasks are grouped according to the service they provide, being offloaded onto the same edge node. In [20], authors design a task scheduling policy that satisfies latency requirements of different users. The proposed mechanism uses task buffering and a heuristic algorithm to decide if the task is executed either locally on the mobile device or remotely offloaded onto a cloud server by calculating the probability of tasks' restrictions accomplishment (third row). In [21] (fourth row), authors formulate the system energy minimization problem as a class of games called congestion games, in which each mobile node is a player and his strategy is to select one of the available nodes to offload its computation. With the aim of minimizing the combined mobile and servers energy consumption, authors prove that the Nash equilibrium always exists in this congestion game formulation. In [22], authors propose a distributed matching algorithm to group tasks from spatially proximate user nodes with mutual task popularity, allowing to cache the tasks' computations to minimize the computation latency (fifth row). Urgaonkar et al. [23] (described in the sixth row of Table I), propose an online workload scheduling using Lyapunov optimization techniques for applications allocation and computation migration, with the aim of minimizing the operational network costs. In [24] (seventh row), authors consider a system formed by one edge computing point and a cloud server to use a two-stage heuristic algorithm based on semidefinite relaxation for tasks allocation and computation migration. The aims of this work are to minimize the energy consumption (focused on the user's mobile) and execution latency.

As Table I shows, none of the existing works (except ours, characterized in the last row of Table I) consider software characteristics of the infrastructure (such as operating system, support for software virtualization, or third party libraries) when deciding the tasks allocation. Additionally, our approach allows more fine-grained computation offloading than binary offloading, supporting parallel execution between the user mobile and the edge nodes. When possible, our approach is able to adapt the applications' tasks according to the existing edge infrastructure, with heterogeneous nodes, and with the objective of minimizing the energy consumption according to the user/infrastructure necessities.

There is proved evidence that SPLs provide benefits to IoT systems regarding the management and modelling of variability. Indeed, there are several works that use a SPL approach to model cyber-physical systems, as we comment below. In most of approaches, variability is managed by FMs [25]. Traditionally, SPLs use single layer FMs [26], in which only application features are contemplated and not the variability of the infrastructure. This forces to assume the specifications of the devices in which the software will be deployed.

However, although the IoT functionalities are typically conditioned by the hardware and software characteristics of the devices in which they are deployed, the infrastructure is often neglected in the SPL models. Typically, when included, the

modelling of infrastructure-specific features is intermingled with the rest of application-specific features. Little, incipient and very specific work [27][28] has been done to model separately those application features that are platform dependent, with the aim of reflecting the possible independent evolution and the restrictions that the infrastructure imposes on the deployment of software applications. Although the separated modelling 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 edge computing environments. This neglect in relation to modelling platform dependencies also limits software infrastructure optimization, as many features (e.g., performance, energy consumption) depend on the devices in which applications run [29][30]. In addition, paradigms such as edge computing have changed the way in which applications are deployed, requiring to manage the devices heterogeneity, something that is not always considered in these approaches.

In contrast, our approach supports the explicit modelling 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.

III. Our approach

Figure 1 shows a general overview of our approach, which encompasses everything from the production of software to its optimal deployment in edge-based infrastructures in four steps. Roughly, in step (1) the developer selects the features of the application's FM that represent the desired application functionality, and this is the *configured application FM*. It also uses the infrastructure FMs (software and hardware) to configure the characteristics of the nodes that compose the edge environment, and this is the *configured infrastructure model*. The FMs used in this step are detailed in section IV. Secondly, the *Task Mapping* module uses the configured application FM to map the selected application features to the tasks interfaces defined to provide such functionalities. The mapping is represented as a task-call graph.

This task-call graph, along with the deployment infrastructure configuration, are the inputs for the *Tasks Implementation Selector* (TIS, tagged with (3) in Figure 1). Each node of the task-call graph obtained in step (2) can be mapped to different task's implementations. Also in (3), each configuration is evaluated (using the module widely explained in Section VI), checking if the application requirements are met, and estimating the energy consumption of the overall set of tasks. This is used to find the task implementation that requires less energy for each application functionality, according to the infrastructure's nodes features. This module also checks the feasibility of the deployment for a given number of users, regarding the initial status of the nodes configured in (1). Note that this status (e.g., available devices, workload, etc.) can change over time, so monitoring the infrastructure is necessary once the tasks are deployed (as depicted in step 4 in Figure 1). As a result, the Tasks Implementation Selector (step 3) returns a second task-call graph of the application using specific tasks' implementations instead of interfaces. The first three steps allow addressing RQ1. In the last step (tagged with (4) in Figure 1), the *Optimal Task Assignment Framework* (OTAF, Section VII) assigns each application task to the most suitable edge node for its execution minimizing the energy consumption and assuring the application QoS, addressing RQ2.

IV. Feature modelling

The first step towards the deployment of an IoT application requires configuring the application and the devices of the infrastructure using FM. Firstly, the software engineer is concerned with the definition of FMs. A FM represents, 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. In our case, the SPL engineer specifies three different FMs: two of them for modelling the infrastructure (one containing the hardware and another with the software characteristics) and a third one with the features of a family of applications of a certain domain (e.g., augmented reality apps). An example of the FM of a family of augmented reality applications is shown on top left corner of Figure 4 and will be explained in Section VIII. Note that all these models are defined only once by the domain engineer, and are reused to deploy different applications of the same family or domain.

As stated in Section II, the separated modelling of the infrastructure from application features is a contribution of our work. Although applications are typically conditioned by the hardware and software characteristics of the devices in which they are deployed, the infrastructure is often neglected in the SPL models, which map the application's features selected from the FM to pieces of software in a 1:1 relation. Instead of this, we propose using two additional FMs (for the hardware and software characteristics respectively, apart from the application one, so three in total) to fully adapt the application to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

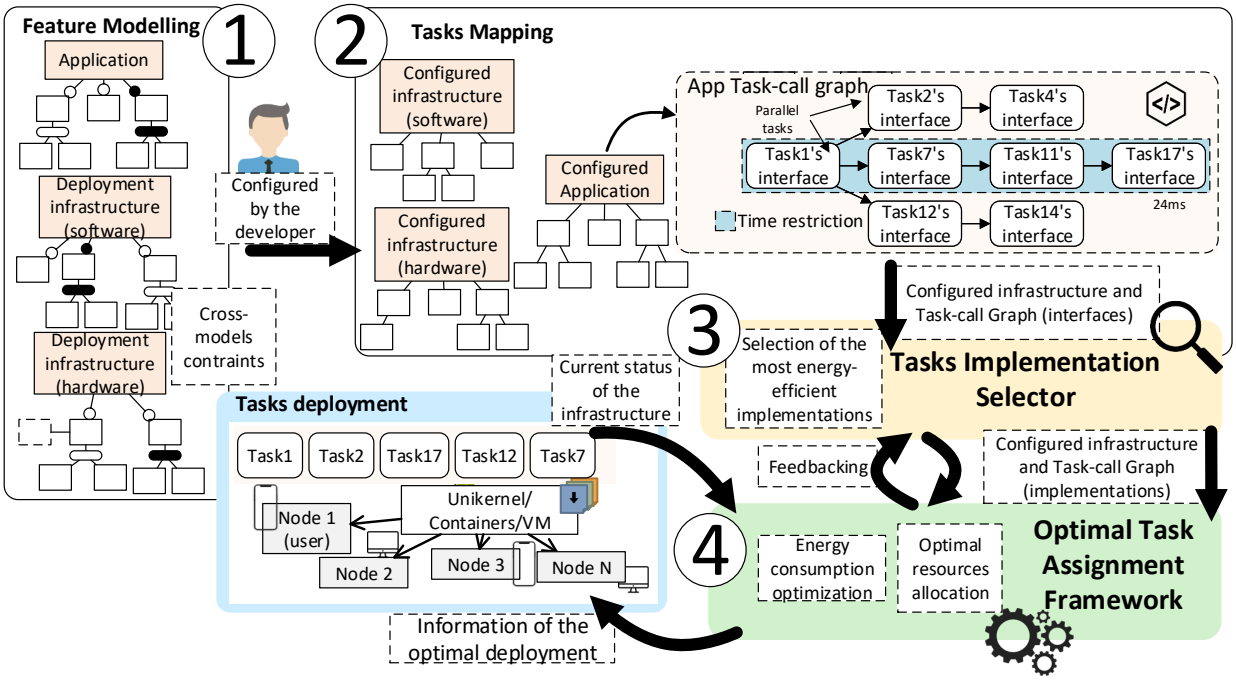

Fig. 1. General overview and capabilities of our approach: (1) configuration of feature models (FMs) with the characteristics of the infrastructure and application; (2) mapping of application's features onto tasks' interfaces; (3) selection of the optimal task implementation for each application interface; (4) optimal task assignment decision minimizing energy consumption.

the infrastructure. The use of multi layer feature models [31] allows us to distinguish between the hardware and software characteristics to reuse and facilitate the configuration of the infrastructure FMs. To avoid developing software products not supported by the infrastructure, the constraints between layers (*cross-models constraints*) are applied to maintain consistency between models [32][27].

Unlike the application FM, which depends on a specific application domain, the infrastructure FMs are extensible and reusable for any IoT applications deployed in the same infrastructure. In this section we focus on the description of these two extensible and reusable FMs for IoT infrastructures.

Nodes are characterized by a set of hardware and software features: type of device, computing capacity, amount of memory, sensing units, network capabilities, operating system, virtualization technologies supported, etc. These features are directly related to the type of tasks that can be deployed on them, that is, the execution requirements of the applications' tasks must meet the node's hardware and software features. Since our FMs contain features that can appear more than once, we use feature models with cardinality [33]. FMs with cardinality allow instantiating the same feature multiple times ([1...*] in Figure 2, e.g., communication capabilities of nodes). Numerical features contain non discrete numerical characteristics of the nodes [34] (e.g., CPU frequency), whose value is included manually by the developer. The FMs presented contain relevant information that will be used to predict the latency and energy consumption behaviour of nodes during the application deployment.

A. *Feature model of the hardware infrastructure*

Computers, mobile devices, Raspberry type computers, IoT Gateways (routers with processing capacity), home appliances, and smart home accessories located on the edge–such as Alexa and Google Home–are some examples of nodes that can be part of a deployment infrastructure. The behaviour of nodes is defined by their role: *computing nodes* (capable to receive tasks offloaded from other nodes) and *interactive nodes* (used directly by the users). For instance, depending on the role of an Alexa device, its function will be to receive instructions from the user (interactive node) and/or execute tasks offloaded from other users/applications (computing node). The user interactions supported by devices are represented by the feature *Capabilities*, which models their data input/output physical channels, such as keyboard, microphone, speakers, camera, and so on.The RAM (Random Access Memory) and HDD (Hard Disk Drive) define the amount of memory and storage of nodes. The computation power is defined by their CPU (Central Processing Unit), and optionally, GPU (Graphics Processing Unit). Technically speaking, CPUs are defined by their number of cores and their frequency (Hz) and GPUs by their RAM and frequency. To be candidates for offloading, devices need at least one network connection, modelled by the typical networking layers (Figure 2); since devices may have several connectivity capabilities, they are modelled as a feature with cardinality. Network connections are also characterized by their upload and download transmission rates (R^{Tx} and R^{Rx} respectively, bits/sec) and P^{Tx} and P^{Rx} (W), that are the upload/download transmission powers (assumed like constants [9][19]), involved in the latency and energy consumption of communications. In case of multihop networks, *nHops* represents the maximal minimal number of hops between

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

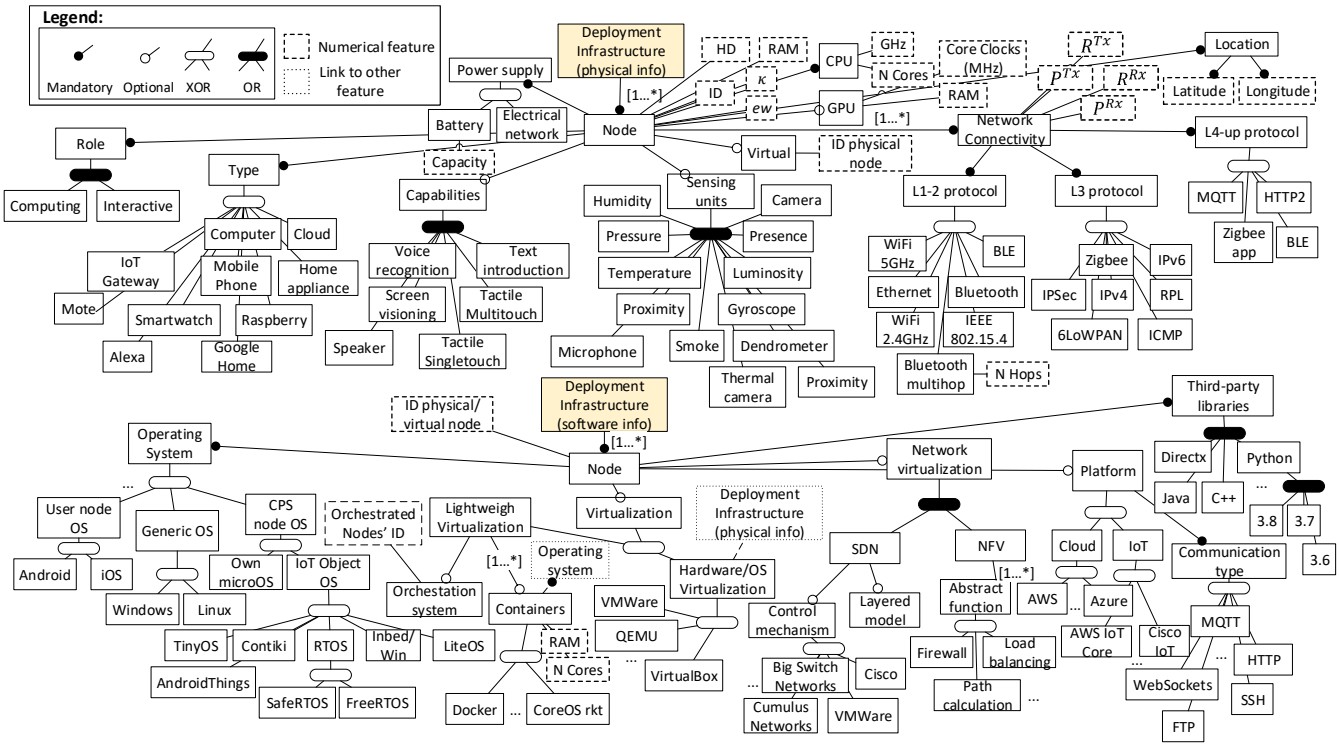

Fig. 2. Infrastructure feature models: physical layer (up) and software layer (bottom)-defined in the step 1 of Figure 1)

nodes. The location of the devices is determined by their latitude and longitude. Optionally, devices may have a set of sensing units associated: temperature, proximity sensors, microphones, or cameras are some examples. Other remarkable features contained in the model are κ, ew and the power supply. κ is the effective switched capacitance depending on the chip architecture in the hardware that directly influences the energy consumption of computing tasks [35][36]. For its part, ew (with a value between 0 and 1) allows to manually set the importance of saving energy in the node. Finally, the power supply determines if the device works plugged into the electrical power or is battery-operated.

B. Feature model of the software infrastructure

Software characteristics of nodes include the operating system and third-party libraries, and optionally, the offloading enabling technology (i.e., virtualization and containment technologies supported) and the platform configuration (for cloud devices). Regarding virtualization, we distinguish between hardware/OS virtualization [37] and lightweight virtualization based on containers [38]. In the first case, software applications run on virtual hardware, allowing to define several machines inside the same device, which are considered as infrastructure's nodes (these are modelled using a link between these features and the hardware FM [39]). These virtual nodes are constrained by the resources of the physical device where they are instantiated. This is managed by adding the identifier of the physical node to the hardware feature model with the physical information. This virtualization feature allows to select the technology used for virtualization with the aim of deploying virtual machines on demand. For its part, container based virtualization emulates an operating system rather than the underlying hardware. As before, we use links to create a relationship between features to configure the operating system of containers, which avoid repeating parts of the FMs. It is possible to define runtime options related to memory, CPUs, and GPUs, limiting its usage. Once again, knowing their technology allows launching new containers on demand. Container technologies (e.g., Docker) allow managing the resources among containers instantiated in the same node according to the containers' workload. Nevertheless, they do not permit workload sharing among containers running on different nodes. This is the aim of orchestration systems for distributed architectures, which are modelled by adding the ID of the nodes subscribed to the orchestration. As for the network connectivity, nodes may have one or more virtual networks associated [40]. The deployment infrastructure may include cloud nodes, that will be described by the platform and communication type in order to adapt the communication of the application tasks to them. This type of nodes can model both services in datacenters in the core of the Internet, and small datacenters in the edge of the Internet (cloudlets), differing in latency and computation characteristics.

V. TASK MAPPING

Once the application and the infrastructure are configured, the *Tasks Mapping* process starts.

In this step, the configuration of the application's feature model obtained in the previous step is mapped with the tasks' interfaces of the application. Each feature of the application's feature model is related to one or more task application's

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

Module 1 Tasks Implementation Selector (TIS)

```
Data: Nodes; possibleConfigurations; timeRestrictions; nUsersToSupport
Result: Min(configurationsConsumption)
1  foreach configuration ∈ possibleConfigurations do
2      Tasks ← configuration;
3      opt = Optimize();
4      opt.add( foreach task ∈ Tasks{
           assignedNode(task) ≠ null } );
5      opt.add( foreach task ∈ Tasks {
           Sum(assignedRAM(task,assignedNode(task))<node_RAM*nUsersToSupport}});
6      opt.add( foreach task ∈ Tasks {
           assignedRAM(task, assignedNode(task)) ≥ task_RAM });
7      opt.add( foreach task_i ∈ Tasks {
           foreach task_j ∈ Tasks {
            if (hasToSendData(task_i,task_j) & assignedNode(task_i) ≠
               assignedNode(task_j)) then {
             connectionUsed(task_i) ≠ null }}} );
8      opt.add( foreach task_i ∈ Tasks {
           foreach task_j ∈ Tasks {
            if (hasToSendData(task_i,task_j) & assignedNode(task_i) ≠
               assignedNode(task_j)) then {
             connectionUsed(task_i) ∈ assignedNode(task_i).connections ∩
               assignedNode(task_j).connections} else {
             connectionUsed(task) = Internet }}} );
9      foreach tr ∈ timeRestrictions do
10         opt.add(Sum[ foreach task_i ∈ tr {
               foreach task_j ∈ tr {
                 timeComputation(task_i, assignedNode(task_i)) + tCommunication(
                  task_i, task_j, assignedNode(task_i), assignedNode( task_j),
                  connectionUsed(task_i) }}] ≤ tr.time);
11     end
12     opt.add( foreach task ∈ Tasks {
           task_requirements ⊆ assignedNode(task)_features } );
13     opt.add(communicationEnergyCost = Sum [
           foreach task_j ∈ Tasks {
            foreach task_i ∈ Tasks {
             EnergyCommunication( task_i,task_j,assignedNode(task_i),
             assignedNode(task_j), connectionUsed( task_i))) }}]);
14     opt.add(computationEnergyCost = Sum [
           foreach task ∈ Tasks {
            energyComputation(task, assignedNode(task)) }]);
15     opt.minimize(communicationEnergyCost + computationEnergyCost);
16     sat = opt.check(); // Checking the satisfacibility
17     if sat then
18         result = opt.solve();
19         configurationsConsumption.add(configuration.id,
             result.communicationEnergyCost + result.computationEnergyCost);
20     end
21 end
22 return(Min(configurationsConsumption));
```

interfaces. At the same time, each application's interface is included into a task-call graph, which is typically a finite directed graph with no directed cycles. The set of vertices represents the application tasks and the edges represent their dependencies. This representation allows to detect tasks whose beginning of execution depends on a previous task (sequential dependency) and the existence of parallel tasks (see top right corner of Figure 1).

Typically, applications have functionalities with time restrictions –e.g., the tasks in charge of screen refreshing in a game respecting its FPS (Frames Per Second). Corresponding interfaces are grouped into sets with sequential dependency, *tr*, where tr_{time} is the maximum time to be completed, being all of these dependencies grouped in the set *timeRestrictions* (Figure 1). These restrictions are specific to the application and must be fulfilled by all users. As a result, this process returns a task-call graph of the application, where the nodes are tasks' interfaces.

VI. Tasks Implementation Selector

At this point, the tasks' interfaces and their timeline are determined. The tasks' interfaces obtained from the previous step can have alternative implementations, which may differ in the requirement of certain hardware (e.g., a device with keyboard) and/or software characteristics (e.g., Java for Android as a third party library), and in the resources consumed or demanded. The aim of the *Tasks Implementation Selector* is to evaluate each implementation alternative for the interfaces, returning the most adequate according to the infrastructure. Managing different implementations of the same task allows: (1) to execute applications regardless of the deployment infrastructure, as it enables adapting the software product's components to available infrastructure; (2) to use implementation optimized to the resources of the target node (e.g., lighter versions of software for battery-powered devices); and, in case of infrastructures composed of several devices, (3) to take advantage of edge computing by selecting the most appropriate device to offload each application's task implementation.

We address the implementation decision problem as a constraint-satisfaction problem that is resolved with a SMT-based approach. SMT (Satisfiability Modulo Theories) is a formalized approach to constraint programming. Formalized as a form of the constraint satisfaction problem, (1) the algorithm of the solver always returns a solution, which guarantees that the deployment is feasible or the impossibility to deploy the application if no solution is found (addressing RQ2); and (2) a large number of constraints (required to solve the problem at hand) help SMT-solvers to reduce the search space and to find the optimal solution faster [41][42]. However, the flexibility of our approach allows to use other mathematical models for optimization.

The characteristics of the nodes modelled in Section 2, along with the ones of the tasks, are used to predict their latency and the energy consumption associated [19][36][9]. The first expression of Equation 1 shows the expression used to calculate the computation time (sec) 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 can be estimated [11]–and the CPU power of node *n* (cycles per second, F_n). The communication time (sec) 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*), plus the propagation delay (s) between *n* and *z* ($t_{n,z}^{prop}$)–assumed as 0 between edge devices [9][19]–as seen in the second expression of Equation 1:

$$Tcomp_{i,n} = x_{i,n}\frac{w_i}{F_n}$$
$$Tcomm_{i,j,n,z} = x_{i,n}h_{i,j}\left(\frac{c_{i,j}}{Min(R_n^{Tx}, R_z^{Rx})} + t_{n,z}^{prop}\right) \quad (1)$$

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 [19].

The energy consumption in the nodes is influenced by several factors, such as the usage of CPU, storage, and RAM, being the CPU usage the most influential [43] and the one in which EC approaches typically base their models [9]. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

first expression of Equation 2 shows the expression used in this work to predict the computation energy consumption (J) required by node *n* to compute task *i* ($Ecomp_{i,n}$). The energy consumption to make task *i* communicate with task *j* (using WLAN or the Internet) is given by the sum of the energy consumption in the sender and receiver nodes (*n* and *z*, respectively), as shown in the second expression of Equation 2 [9]:

$$Ecomp_{i,n} = x_{i,n}\kappa_n w_i F_n^2 ew_n$$
$$Ecomm_{i,j,n,z} = x_{i,n}h_{i,j}P_n^{Tx}\frac{c_{i,j}}{R_n^{Tx}}ew_n + x_{j,z}h_{i,j}P_z^{Rx}\frac{c_{i,j}}{R_z^{Rx}}ew_z \quad (2)$$

For the sake of simplicity, we define several functions:

- timeCommunication ($task_i$, $task_j$, $node_n$, $node_z$, CONNECTION_TYPE): time required (s) in nodes *n* (sender) and *z* (receiver) to send/receive the output/input data of tasks i and j using *CONNECTION_TYPE*;
- timeComputation (task, node): execution time (s) to compute *task* in *node*;
- energyCommnunication ($task_i$, $task_j$, $node_n$, $node_z$ CONNECTION_TYPE): energy consumption (J) in nodes *n* (sender) and *z* (receiver) to send/receive the output/input data of tasks i and j using *CONNECTION_TYPE*;
- energyComputation (task, node): energy consumption (J) in *node* to compute *task*;
- hasToSendData ($task_i$, $task_j$): true if $task_i$ transmits data to $task_j$;
- assignedNode(*task*): returns the node assigned to *task*;
- assignedRAM(*task*, *node*): RAM (Mb) allocated in *node* to *task*;
- connectionUsed(*task*): returns the connection used to send the data of *task*.

Module 1 shows the pseudo-code of the Tasks Implementation Selector. The module receives as input the information of the nodes (modelled as explained in Section V), the task-call graph of interfaces (with the time restrictions) and the number of users to support (*nUsersToSupport*). From lines 1 up to 21, it iterates with each configuration possibility. Concretely, Module 1 does the following: for each set of tasks, line 4 assures that tasks are assigned to an unique node, while lines 5 and 6 check that nodes allocate sufficient RAM to execute each task and have enough resources to support them (*nUsersToSupports*). Line 7 guarantees that tasks that need to send data have a connection associated, while line 8 assures that for interconnected tasks, the sending and receiver nodes have the selected connection among their connectivity's capacities. Lines 9 to 11 check that the time restrictions of the application are accomplished, while line 12 guarantees that nodes meet the tasks requirements. Lines 13 and 14 determine the energy consumption associated to tasks computation and communication. Line 15 asks for the solution that reduces the energy consumption the most. Lines 16 to 20 check the satisfiability of the problem and include the energy consumption in a dictionary with pairs *<id, energy consumption>* if positive. Finally, at the end of the loop (once all the configuration possibilities have been checked), Module 1 returns the configuration with the minimal energy consumption.

Module 2 Optimal Task Assignment Framework (OTAF)

Data: Nodes; Tasks; timeRestrictions; nUsersToSupport
Result: assignedNode(Tasks); RAMassigned(Tasks, Nodes); connectionUsed(Tasks)
1 Nodes ← currentStatus;
// Parameters to optimize:
2 assignedNode(*task*);// Optimal node to execute *task*
3 assignedRAM(*task*, *node*);// RAM allocated in *node* to *task*
4 connectionUsed(*task*)// connection used to send the data of *task*
5 opt = Optimize();
6 opt.add(**foreach** task ∈ Tasks{
assignedNode(task) ≠ null });
7 opt.add(**foreach** task ∈ Tasks {
Sum(assignedRAM(task,assignedNode(task))<$node_{RAM}$*nUsersToSupport}});
8 opt.add(**foreach** task ∈ Tasks {
assignedRAM(task, assignedNode(task)) ≥ $task_{RAM}$ });
9 opt.add(**foreach** $task_i$ ∈ Tasks {
foreach $task_j$ ∈ Tasks {
if (hasToSendData($task_i$,$task_j$) & assignedNode($task_i$) ≠
assignedNode($task_j$)) **then** {
connectionUsed($task_i$) ≠ null }}});
10 opt.add(**foreach** $task_i$ ∈ Tasks {
foreach $task_j$ ∈ Tasks {
if (hasToSendData($task_i$,$task_j$) & assignedNode($task_i$) ≠
assignedNode($task_j$)) **then** {
connectionUsed($task_i$) ∈ assignedNode($task_i$).connections ∩
assignedNode($task_j$).connections} **else** {
connectionUsed(task) = Internet }}});
11 **foreach** *tr* ∈ *timeRestrictions* **do**
12 opt.add(Sum[**foreach** $task_i$ ∈ tr {
foreach $task_j$ ∈ tr {
timeComputation($task_i$, assignedNode($task_i$)) + tCommunication(
$task_i$, $task_j$, assignedNode($task_i$), assignedNode($task_j$),
connectionUsed($task_i$) }}] ≤ $tr.time$);
13 **end**
14 opt.add(**foreach** task ∈ Tasks {
$task_{requirements} \subseteq assignedNode(task)_{features}$ });
15 opt.add(communicationEnergyCost = Sum [
foreach $task_j$ ∈ Tasks {
foreach $task_i$ ∈ Tasks {
EnergyCommunication($task_i$,$task_j$,assignedNode($task_i$),
assignedNode($task_j$), connectionUsed($task_i$))) }}]);
16 opt.add(computationEnergyCost = Sum [
foreach task ∈ Tasks {
energyComputation(task, assignedNode(task)) }]);
17 opt.minimize(communicationEnergyCost + computationEnergyCost);
18 sat = opt.check(); // Checking the satisfacibility
19 **if** *sat* **then**
20 solution = opt.solve();
21 **end**
22 **return(solution);**

VII. Optimal Task Assignment Framework

This module takes as input the output of the *Tasks' Implementation Selector* to select the device that best fit the execution of each task (addressing RQ2). This decision considers the current status of each node of the infrastructure.

As in the previous section, we use a SMT solver (for the same reasons) to obtain the solution for the decision problem of Module 2. The pseudo-code of this component is quite similar to Module 1, as the constraints to meet are the same. Concretely, instructions presented from lines 5-18 in Module 2 coincide with the ones contained in Module 1 from lines 3-16– if *nUsersToSupport* is not received as an input, it is considered as 1 and the OTAF will be launched each time a new user joins to the service. Considering their coincidences, this time we focus on the node assigned to each task, RAM allocated, and the connection used. Additionally, Module 2 checks the current status of the nodes at the beginning of its execution in order to evaluate the feasibility of the deployment (line 1). As Module 2 shows, unlike in Module 1, the body of the problem

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

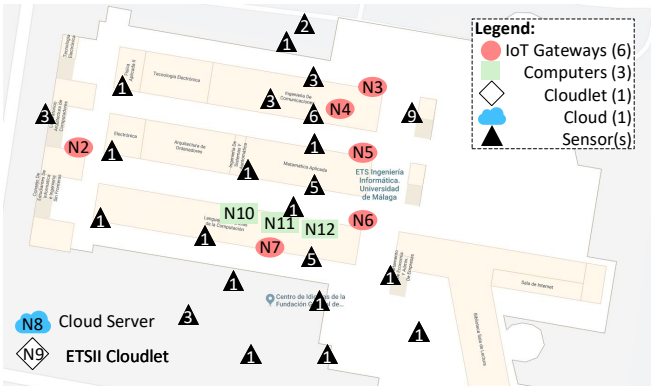

Fig. 3. Geographical location of the devices in the School of Computer Science Engineering buildings

is executed only once. Finally, the solution of the OTAF is used to deploy the application in the infrastructure.

Edge nodes can provide computation services for multiple tasks from multiple users simultaneously using processor sharing. We assume that the computing capacity of the nodes do not change during the processing of one task but can change across tasks [16]. The same occurs with the transmission rates [19]. Note that although in this paper we focus on the OTAF taking advantage of EC to minimize the energy consumption, it can be used to minimize the latency or even to return a trade-off between both (multi objective optimizations) [44], as we manage enough information to do so.

VIII. Evaluation

This section evaluates our approach, studying the reduction of power consumption obtained by task offloading and measuring the execution time to find a task offloading solution for different problem sizes.

A. Reduction of energy consumption

To evaluate the reduction in energy consumption, we apply our approach to a real IoT infrastructure installed at the School of Computer Science Engineering of the Universidad de Málaga. The goal is to study to what extent our approach reduces energy consumption by distributing the computational load among edge/cloud nodes in a real IoT scenario, considering this infrastructure is shared by several applications at the same time. This infrastructure has a large number of physical sensor motes distributed throughout the four buildings of the school, which send sensing measures' information to six Meshlium IoT gateways. The IoT gateways periodically send the information collected from the sensors to a cloudlet, which updates its databases with up-to-date measures. The infrastructure also contains 3 computers acting as edge nodes, and a cloud server. Figure 3 shows the geographical location of the devices on a map of the School of Computer Science Engineering buildings. Our approach benefits from the spare computational power of these devices. The deployed mobile application (app for short) is named 'La Universidad Aumentada'. This app, developed by the Universidad de Málaga, uses augmented reality to show motivational messages from successful students. These messages are visible by scanning QR (Quick Response) codes spread over the university. This app gathers the measurement data from the environment collected by the aforementioned IoT sensors distributed throughout the physical campus. These data allow the augmented reality app to enrich the information presented to the user considering their location. Then, depending on the user location determined by the GPS (Global Positioning System) of the user mobile device, the app generates perceptual information with data from the nearby IoT sensors (e.g., temperature, humidity, presence, etc). Augmented reality applications are widely used in EC approaches due to their inherent collaborative properties in terms of data collection in the uplink, computing at the edge, and data delivery in the downlink. In addition, augmented reality apps are computational-intensive and delay-sensitive, and processing all the information on mobile devices is generally prohibitive because it would impact users' expectations in terms of battery lifetime [45].

A simplified version of the application's feature model is shown at the top left corner of Figure 4, while the configured FM for 'La Universidad Aumentada' application is on top right corner of the same figure. Depending on the characteristics selected, the application will have different QoS requirements. For instance, the characteristic FPS determines the maximum time to complete the tasks in charge of processing and refreshing images (30 per second selected, i.e., 34 ms per execution–time restriction 1 in Figure 4). The resulting task-call graph of implementations (once processed by the *Tasks Implementation Selector*, composed by 10 different tasks) is shown at the bottom of Figure 4. In this computing model, the functionality of the app is separated into tasks related to data processing and storage capability. The workload of mobile devices can be reduced by offloading the computation intensive tasks that deal with the generation of augmented information to near nodes in the edge network.

For our experiments, the characteristics of the 12 nodes (the user node, 10 edge nodes and 1 cloud node) have been randomly generated. Concretely, the CPU speed of IoT gateways ranges from 1 to 1.6 GHz, the cloud server from 2.4 to 4 GHz, the cloudlet from 2 to 2.4 GHz, and the rest of edge devices from 1.6 to 2 GHz.

R^{Tx} and R^{Rx} have been set between 100 and 150 Mbps for edge devices and from 8 to 10 Mbps for cloud devices. The propagation delay between the cloud device and the rest of nodes has been set from 0.02 to 0.1 s. κ has a value between $1 \cdot 10^{-9}$ and $1 \cdot 10^{-11}$ [46][19]. P^{Tx} and P^{Rx} have values between 1 and 1.5 W in all cases [19]. Finally, *ew* has been set to 1 for all nodes. To avoid the mobility problem in edge computing [47], the task offloading process relies on WLAN and the Internet for data transmission.

Experiments consider two scenarios. In the first one (Scenario 1), the nodes reserve a fixed amount of resources (2 GB of RAM and one CPU core) dedicated to computation of offloaded tasks. In the second one (Scenario 2), the nodes do not reserve specific resources for offloaded tasks, so the feasibility of the task offloading process and the reduction in energy consumption will depend on the current nodes workload (randomized in each experiment from 0 to their maximal capabilities). Each test is performed 30 times.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

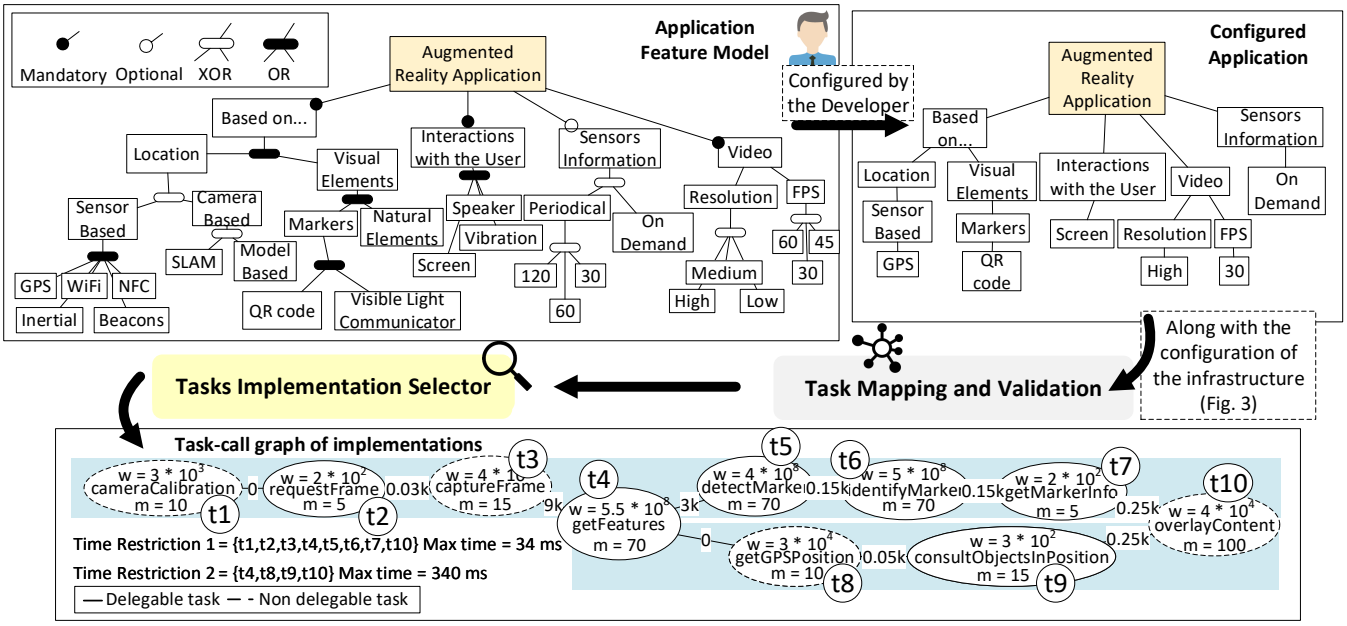

Fig. 4. Feature model of the application (top left corner), application configuration (top right corner), and task-call graph of tasks' implementations (bottom)

Table II shows the reduction of energy consumption (REC) obtained in the experiments performed for the case study described in Figure 4 for both scenarios. For Scenario 1, three rows detail the task offloading solutions (distribution of tasks in the different nodes, in columns N1 to N12, and RAM used) and the reduction of energy consumption (REC) for three different states of workload in the infrastructure, which depends on the number of users: users $\leq$ 13; 14 $\leq$ users $\leq$ 26; 27 $\leq$ users $\leq$ 39.

These results allow to compare the energy consumption of distributing the application according to the assignment solution obtained by our approach with the energy consumption of running the entire application on the user device. This reduction in the energy consumption (columns REC in Table II) is given as a percentage (%). The reduction is calculated considering the energy consumption of all nodes of the infrastructure (including the user one) and considering just the energy consumed in the user node–columns named *REC (all nodes)* and *REC (user node)* respectively.

The aim of the first scenario is to illustrate how the reduction in energy consumption can be affected by the nodes availability, and how our solution adapts consequently. The first scenario considers three different states of node availability. Firstly, all nodes are fully available, so the OTAF assigns tasks to the nodes which consume less energy and are capable of running the tasks while satisfying the QoS, assigning the tasks *t2, t5, t6,* and *t7* to node *N9* and *t9* to *N11*. This tasks assignment achieves a 48% reduction in power consumption considering all nodes (column 16 of Table II). After attending the offloading requirements of 13 users, *N9* cannot allocate more tasks and the infrastructure goes to State 2, being *N9* unavailable to allocate tasks. This time, *N12* is the selected node to deploy the tasks previously assigned to *N9*, while *N11* continues having task *t9* assigned, obtaining a 43.5% reduction in the energy consumption (considering all nodes). Once again, after serving another 13 users (26 users in total), *N12* cannot run more offloaded tasks, and the infrastructure goes to State 3. This time, both nodes *N9* and *N12* are busy, and the most suitable node to run *t2, t5, t6,* and *t7* is now *N7*. The reduction in the energy consumption, considering the consumption of all nodes, is 37.3%. Despite the difference in the number of available nodes and tasks allocation, for the three states the user node runs the same tasks and consumes the same amount of energy, reducing its energy consumption by 62.2% compared to an offline execution (last column of Table II).

The second scenario refers to a deployment infrastructure shared by many users and applications in which resources can not be reserved for the execution of specific applications. The results are provided in row *Non-fixed resource allocation* in Table II. In this case, the CPU workload and the free RAM of each node are also randomly set in each experiment. For this scenario, the results show the reduction of energy consumption obtained, instead of the assignment itself. Last row of Table II shows the average, minimum, maximum, and standard deviation of the reduction in the energy consumption obtained, both considering all nodes and taking into account only the user node. Notice that, for a randomly generated nodes status, the average reduction obtained in the energy consumption when all nodes are considered is 41.1%, being 55.2% the maximum value obtained and 33.6% the minimum, with a standard deviation of 9.3%. When only the energy consumption in the user node is considered, we obtain an average reduction of 56.5%, 65.1% as maximum and 41.2% as minimum, being the standard deviation of 8.8%. The reduction in energy consumption obtained in the second scenario has decreased as compared to the reduction observed in the first one. The reason is that some infrastructure's nodes may not be available for task offloading, so the OTAF adapts the solution to the available resources, and, as a result, some tasks

TABLE II
REDUCTION IN THE ENERGY CONSUMPTION (%) OF OUR TASK ASSIGNMENT SOLUTION VS RUNNING THE ENTIRE APPLICATION IN THE USER DEVICE, FOR THE CASE OF FIGURE 4

	Nodes:		N1 (user node)	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	REC[a] (all nodes)	REC (user node)
Scenario 1: Fixed resources allocation	users $\leq$ 13 (State 1)	Tasks assigned	t1,t2,t3,t8,t10	-	-	-	-	-	-	-	t2,t5,t6,t7	-	t9	-	**48.0%**	**62.2%**
		RAM (Mb)	205	-	-	-	-	-	-	-	150	-	15	-		
	14 $\leq$ users $\leq$ 26 (State 2)	Tasks assigned	t1,t2,t3,t8,t10	-	-	-	-	-	-	-	busy	-	t9	t2,t5,t6,t7	**43.5%**	**62.2%**
		RAM (Mb)	205	-	-	-	-	-	-	-	-	-	15	150		
	27 $\leq$ users $\leq$ 39 (State 3)	Tasks assigned	t1,t2,t3,t8,t10	-	-	-	-	-	t2,t5,t6,t7	-	busy	-	t9	busy	**37.3%**	**62.2%**
		RAM (Mb)	205	-	-	-	-	-	150	-	-	-	15	-		
Scenario 2: Non-fixed resources allocation	**REC considering all nodes(Avg/Max/Min/Std)**										**41.1 / 55.2 / 33.6 / 9.3 %**					
	REC in the user node (Avg/Max/Min/Std)										**56.5 / 65.1 / 41.2 / 8.8 %**					

[a] REC: Reduction in the energy consumption

cannot be offloaded. Although this can be considered the worst scenario for deployment, even in this case the OTAF finds an offloading solution that satisfies the user node.

B. Scalability

The time needed by our modules to provide a solution varies according to the size of the problem [48]. This section evaluates their execution time for different problem sizes. With this purpose, we develop a *Benchmark*[1] version of the modules, which allows setting the number of devices, tasks and configuration in the case of Module 1 and devices and tasks in the case of Module 2. The characteristics of the devices (CPU, RAM, workload, etc.) and tasks (connections, computational load, hardware and software requirements, time restrictions, etc.) have been randomly generated in each experiment, and each experiment has been performed 30 times on one thread of an AMD Ryzen 7 1700X processor.

Table III shows the average, maximum, minimum, and standard deviation of the time (*s*) taken by our modules to return a solution for different problem sizes. The average time is graphically represented in Figure 5. The execution time values of Module 1 (TIS, grouped in the first row of Table III) are shown in the left graph of figure 5. The number of tasks and nodes has been set in 20, while the number of configurations has been incremented up to 45. Experiments show that Module 1 requires around 85 seconds to find the most suitable set of implementations to deploy an application with 10 different configurations. This time increments up to 430 seconds (i.e., 7 minutes and 10 seconds) in case of 45 alternatives of configuration. Note that Module 1 is only executed once, unless changes in the infrastructure affect the deployment feasibility (in this case, Module 2 will not be able to find a solution).

In case of the OTAF (Module 2, grouped in the second row of Table III), the number of nodes has been set in 20, while the number of tasks has been incremented up to 90. The execution time values of Module 2 (OTAF) are shown in the right graph of Figure 5. As expected, the required average time increases in relation to the number of tasks (see Table III). Nevertheless, the difference between maximum and minimum values, as well as the standard deviations, shows that factors like tasks constraints and characteristics of the devices and tasks may affect the problem complexity, increasing the execution time. Results show that, for applications formed by 10 tasks and a heterogeneous infrastructure composed by 20 different devices, the OTAF requires around 2 seconds to obtain a solution, while applications formed by 20 tasks take around 10 seconds and applications formed by 30 take 25 seconds (it is feasible to run the OTAF when the user launches the application, adapting its behaviour once the OTAF returns the solution in consequence [49]). In the worst case of a very granulated application of 90 tasks and 20 devices ($1.23 \cdot 10^{117}$ assignment possibilities) the OTAF has required about 280 seconds (i.e. 4 minutes and 40 seconds). For very partitioned applications, an infrastructure with a fixed amount of resources for task offloading (first scenario of Section VIII) allows having a task offloading solution before launching the application (by setting the value of the number of users to support, which the OTAF receives as an input). Taking all this into consideration, we conclude that the OTAF is capable of providing a solution in a reasonable amount of time.

[1] Source code available at: https://doi.org/10.5281/zenodo.4068123

IX. CONCLUSIONS AND FUTURE WORK

Edge computing expands the deployment infrastructure of mobile IoT applications, providing more sustainable solutions by means of offloading computation tasks to nearby devices. However, the heterogeneity of the devices, the requirements of applications, and the sharing of software and hardware infrastructure can difficult this process.

In this paper, we propose an offloading process that applies SPL to cope with variability of applications and infrastructures in the task offloading decision, considering hardware and software (typically disregarded in code offloading approaches) nodes characteristics. Task offloading is supported by two modules, which firstly adapt the application task implementations to the infrastructure capabilities and secondly assign application tasks to nodes in order to minimize the energy consumption while assuring the QoS. Experiments show that our approach reduces by approximately 41%-62% the energy consumption in the user node, and by 34%-48% the energy consumption of the devices involved in a task offloading solution. The execution time of our modules for different problem sizes has been also evaluated using a Benchmark, and the conclusion is that our proposal returns a solution in a reasonable amount of time.

As future work, we are planning to assign computational power to each task, controlling this parameter in practice using dynamic voltage and frequency scaling (DVFS) techniques [50].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2020.3030197, IEEE Internet of Things Journal

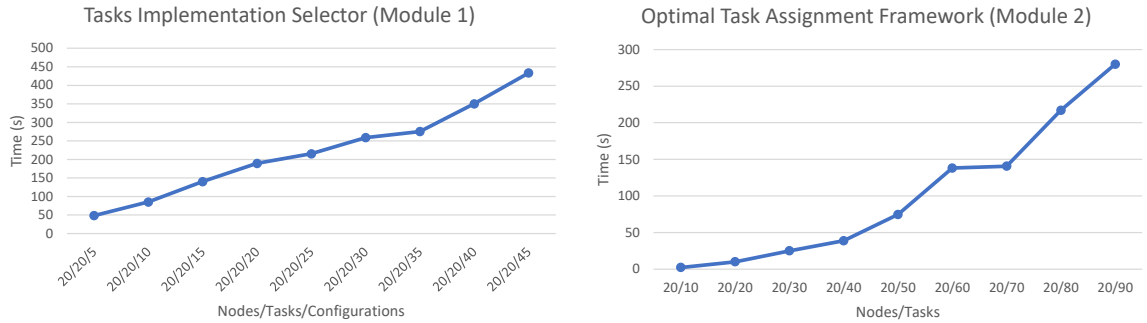

Fig. 5. Graphical representation of the mean values of Table III

TABLE III
EXECUTION TIME FOR MODULES 1 AND 2

N/T/C N/T[a]		20/20/5 20/10	20/20/10 20/20	20/20/15 20/30	20/20/20 20/40	20/20/25 20/50	20/20/30 20/60	20/20/35 20/70	20/20/40 20/80	20/20/45 20/90
TIS[b]	**Mean (s)**	48.51	85.24	140.33	189.50	215.46	258.86	275.46	350.09	433.59
	Max (s)	53.84	91.37	152.92	220.06	243.72	294.36	304.22	393.54	472.68
	Min (s)	37.41	72.11	123.94	164.67	184.04	242.31	256.52	327.58	409.90
	Std	6.44	7.56	13.14	23.73	25.83	20.48	18.28	25.89	23.86
OTAF[c]	**Mean (s)**	2.03	10.04	25.04	38.71	74.57	138.07	140.65	216.98	279.96
	Max (s)	2.31	11.03	27.12	44.39	88.61	156.25	150.44	258.04	301.53
	Min (s)	1.83	8.68	20.04	33.51	60.54	110.50	132.11	175.93	257.96
	Std	0.21	1.01	3.37	5.45	19.84	19.45	9.23	18.05	21.78

[a] N: Nodes. T: Tasks. C: Configurations.
[b] Tasks Implementation Selector - Module 1
[c] Optimal Task Assignment Framework - Module 2

ACKNOWLEDGMENTS

This work is supported by the projects TASOVA MCIU-AEI TIN2017-90644-REDT, MEDEA RTI2018-099213-B-I00 (co-funded by FEDER funds), LEIA UMA18-FEDERJA-157 (co-funded by FEDER funds) and RHEA P18-FR-1081 (MCI/AEI/FEDER, UE).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 363–369.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[4] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, April 2018.

[5] H. Elazhary, "Internet of Things (IoT), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *Journal of Network and Computer Applications*, vol. 128, pp. 105–140, Nov 2018.

[6] S. Bagchi, M.-B. Siddiqui, P. Wood, and H. Zhang, "Dependability in edge computing," *Commun. ACM*, vol. 63, no. 1, p. 58–66, Dec. 2020. [Online]. Available: https://doi.org/10.1145/3362068

[7] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A cloud–MEC collaborative task offloading scheme with service orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2020.

[8] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[10] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[11] S. Melendez and M. P. McGarry, "Computation offloading decisions for reducing completion time," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2017, pp. 160–164.

[12] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[13] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 2:1–2:23, Feb. 2019. [Online]. Available: http://doi.acm.org/10.1145/3284387

[14] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278 – 289, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18301973

[15] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[16] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, Nov 2017.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct 2015.

[18] L. Fuentes, "Variability variations in cyber-physical systems (keynote)," in *Software Architecture-13th European Conference, ECSA 2019, Paris, France, September 9-13, 2019, Proceedings*. Springer, 2019, pp. xvi–xvii. [Online]. Available: https://doi.org/10.1007/978-3-030-29983-5

[19] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, Aug 2017.

[20] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing," in *IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6.

[21] Y. Ge, Y. Zhang, Q. Qiu, and Y. Lu, "A game theoretic resource allocation for overall energy minimization in mobile cloud computing system," in *ISLPED'12-Proceedings of the International Symposium on Low Power Electronics and Design*, Sep. 2012, pp. 279–284.

[22] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing

in latency-constrained fog networks," in *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–6.

[23] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015, special Issue: Performance 2015.

[24] M. Chen, M. Dong, and B. Liang, "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)*, March 2016, pp. 3516–3520.

[25] R. T. Geraldi, S. Reinehr, and A. Malucelli, "Software product line applied to the Internet of Things: A systematic literature review," *Information and Software Technology*, vol. 124, p. 106293, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584920300434

[26] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.

[27] M. Lettner, J. Rodas, J. A. Galindo, and D. Benavides, "Automated analysis of two-layered feature models with feature attributes," *Journal of Computer Languages*, vol. 51, pp. 154–172, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1045926X18302362

[28] E. Farahani and J. Habibi, "Feature model configuration based on two-layer modelling in software product lines," *International Journal of Electrical and Computer Engineering*, vol. 9, pp. 1–11, March 2019.

[29] A. Abbas, I. Farah Siddiqui, S. U. Lee, A. Kashif Bashir, W. Ejaz, and N. M. F. Qureshi, "Multi-objective optimum solutions for IoT-based feature models of software product line," *IEEE Access*, vol. 6, pp. 12 228–12 239, 2018.

[30] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2208–2221, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121211001518

[31] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.

[32] J. A. Galindo, D. Dhungana, R. Rabiser, D. Benavides, G. Botterweck, and P. Grünbacher, "Supporting distributed product configuration by integrating heterogeneous variability modeling approaches," *Information and Software Technology*, vol. 62, pp. 78 – 100, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584915000312

[33] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[34] R. Capilla and J. C. Dueñas, "Modelling variability with features in distributed architectures," in *Software Product-Family Engineering*, F. van der Linden, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 319–329.

[35] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 190–194.

[36] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[37] J. Smith and R. Nair, *Virtual machines: versatile platforms for systems and processes*. Elsevier, 2005.

[38] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, Inc., 2015.

[39] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *Software Product Lines*, R. L. Nord, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 266–283.

[40] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[41] A. Niewiadomski, J. Skaruz, W. Penczek, M. Szreter, and M. Jarocki, "SMT versus genetic and OpenOpt algorithms: Concrete planning in the PlanICS framework," *Fundamenta Informaticae*, vol. 135, pp. 451–466, 01 2014.

[42] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz - an optimizing SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 194–199.

[43] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, Jan 2019.

[44] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proceedings of the Theory and Practice of Software, 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Springer-Verlag, 2008, pp. 337–340.

[45] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, June 2017.

[46] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.

[47] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[48] C. Sundermann, T. Thüm, and I. Schaefer, "Evaluating #SAT solvers on industrial feature models," in *Proceedings of the 14th Int. Working Conf. on Variability Modelling of Software-Intensive Systems*, ser. VAMOS'20, 2020. [Online]. Available: https://doi.org/10.1145/3377024.3377025

[49] A. Cañete, J.-M. Horcas, I. Ayala, and L. Fuentes, "Energy efficient adaptation engines for android applications," *Information and Software Technology*, vol. 118, p. 106220, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584919302307

[50] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677–1690, 2011.

Angel Cañete received his MSc degree in Computer Science from the Universidad de Málaga (Spain) in 2017. He is a member of the CAOSD research group, where he participates in several international and national research projects. His research topics include Edge Computing, IoT, Software Product Lines, self-adaptive software and energy efficiency.

Mercedes Amor received her MSc and PhD degrees in Computer Science from the Universidad de Málaga (Spain) in 2005, where she is an Associated Professor (previously, Lecturer from 2001). She is a member of the CAOSD research group. Her research interest mainly deals with self-adaptation of future Internet applications, software architectures, Software Product Lines, Agent-Oriented Software Engineering and Aspect-Oriented Software Development.

Lidia Fuentes received her MSc degree and a PhD in Computer Science, from the Universidad de Málaga. She has done all her teaching work at the Department Lenguajes y Ciencias de la Computación since 1993, being the first female Full Professor of this department. She is the head of the CAOSD research group (http://caosd.lcc.uma.es), and co-authored more than two hundred publications in software engineering techniques applied to IoT and cyber-physical systems. She has an important international profile leading European projects and as a member of program committees of prestigious international conferences. Examples are ECOOP, SPLC, Modularity/AOSD and OOPSLA. She is currently the most cited female scientific of the University of Málaga.