# A Middleware for Mobile Edge Computing

A. Carrega
DITEN – University of Genoa, Italy
Email: alessandro.carrega@unige.it

M. Repetto
CNIT – RU of Genoa, Italy
Email: matteo.repetto@cnit.it

P. Gouvas and A. Zafeiropoulos
Ubitech Ltd, Greece
Email: {pgouvas, azafeiropoulos}@ubitech.eu

***Abstract*—Telecom operators have recently started to deploy massive computing and storage resources at the very edge of their access networks, hence evolving their infrastructures into large, distributed, and capillary computing environments, capable of placing applications very close to users and terminals and well-suited to effectively fulfill the challenging requirements for delay-sensitive applications.**

***Edge computing* is the emerging paradigm that pervasively brings computing in the environment, by shifting processing from data centers to the network edge, allowing a large class of applications in growing fields, like Big Data and the Internet of Things, to be deployed in a very effective way.**

**In this paper, we propose middleware for running applications over heterogeneous environments, made of telecom networks and legacy data centers. We briefly review an emerging architecture for edge computing and describe an integrated solution for developing and deploying modular applications in an automatic way.**

***Index Terms*—Distributed cloud computing, mobile edge computing, distributed software development and deployment**

## I. Introduction

Even though the cloud paradigm is widely adopted in large installations worldwide, it still falls short of supporting latency-sensitive applications that require proximity to "things" in the physical environment (devices, sensors, actuators, gateways, ...). Augmented reality, gaming, video streaming, connected vehicles, and electrical grid protection are just a few examples of the emerging wave of Internet deployments that require mobility support and geo-distribution, in addition to location awareness and low latency.

*Cloud federation* may collect heterogeneous infrastructures that potentially span the entire globe; however, interaction with devices and terminals in the physical environment is still mediated by the Internet and public telecommunication networks, where topologies and typical architectures today in use introduce non-negligible latency, jitter, and bandwidth limitations.

Instead, the concept of *fog computing* envisions "a system-level horizontal architecture that distributes computing, storage, control and networking functions closer to the users along the cloud-to-thing continuum" [1]; the term "fog" was just coined to recall the idea of "a cloud close to the ground" [2]. The cloud-to-thing continuum is a heterogeneous environment that encompasses legacy clouds, edge installations in telecommunication access networks, and smart things. *Edge* (or *mist*) *computing* is a similar concept that only includes clouds and peripheral network installations; it exploits the rich programmable fabric (including computing, networking, and storage resources) that telecom operators are increasingly deploying in the access segment of 5G telecommunication networks [3]. Edge computing provides a highly distributed, pervasive and virtualized platform, by stretching the cloud till the very edge of the network; it is not conceived as full Cloud replacement, rather as an extension to effectively support a new breed of applications and services. Edge computing promises additional opportunities with respect to plain cloud federation, mainly given by *i)* more pervasive and capillary coverage (hence supporting in an effective way distributed low-latency applications); *ii)* the availability of specific information and data from telecommunication infrastructures (e.g., geo-localization, mobility, network status); and *iii)* support for code offloading from smart things and peripheral devices.

Beyond the abstraction of the underlying hardware, the big challenge will be the capability to dynamically compose, self-provision, and manage applications over a large, distributed and heterogeneous infrastructure [4], [5]. Middleware is required between the virtualization layer and the applications that hides the complexity and heterogeneity of the underlying environment and eases the design, development, deployment, and management of distributed applications and services.

The main contribution of this paper consists in concrete design, implementation and preliminary evaluation of edge computing. The base framework has been developed in the context of the ARCADIA project[1]: it entails development, deployment, orchestration, and lifecycle management of distributed applications over cross-domain infrastructures, hence fitting well the structure of edge computing. This framework employees a multi-tier context model in the software development phase, which describes the application and its requirements, to automate orchestration, deployment, and lifecycle management.

The paper is organized as follows. Section II reviews background and related work in the field of pervasive computing. Section III discusses an emerging architecture for edge computing, to outline specific requirements to be fulfilled. Section IV describes the overall structure of the framework to develop and deploy distributed applications, while Section V explains

The final publication is available at IEEE Xplore via https://doi.org/10.1109/MCC.2017.3791021



[1]A Novel Reconfigurable By Design Highly Distributed Applications Development Paradigm Over Programmable Infrastructure (ARCADIA), http://www.arcadia-framework.eu.

why and how this framework implements the architecture for edge computing described in Section III. Section VI discusses implementation and provides preliminary evaluation results. Finally, we give our conclusions and identify open issues and research challenges in Section VII.

## II. Background and related work

A growing number of applications are requesting interaction and access to smart connected devices (Internet of Things). Similarly, many factors (power consumption, size, cost) are pushing to offload complex tasks from Things to data centers [6], [7]. The Cloud paradigm permits flexible and elastic deployment of such kind of applications, but it usually virtualizes a single or a few huge data centers only. This fact definitively makes them "far" (in terms of latency) from users, physical devices, and other installations, hence making hard an effective and efficient interaction between the cloud and the physical environment [8].

The need for closer *proximity* has pushed the design of large-scale, robust and redundant computing systems, as *distributed* and *federated* clouds, where different data centers are jointly operated under a common interface. These models work for coarse-grained localization, e.g., by deploying a different data center for each continent, country, or region (as already done by some big cloud providers), but are not suitable for worldwide capillary coverage.

Some recent proposals already envision the possibility to build capillary clouds by exploiting telco infrastructures, for example for virtualization and offloading[2] [4], [6], [7], [9], [10]. As a matter of fact, Telco's business is undergoing a fundamental shift, and operators are transforming their networks to meet new business challenges. In [11], the authors describe the transformation of traditional wireline, wireless, voice, text, data, and web services to common compute cloud infrastructures (known as "Telco Cloud"). Cloud infrastructures can be set-up on-premise, off-premise, or hybrid, based on SLA, security, and maturity of services. Instead, the work in [12] describes how to enable telco infrastructures to adopt this new paradigm; the technologies proposed allow a Telco to deploy and manage services and relative functions in a distributed cloud infrastructure.

Though the broader concept of fog computing is still in its infancy [1], the definition of edge computing has already made important steps [4]. Edge-computing architectures designed in [9] and [10] envisage the interconnection of micro-installations at the network edge and data centers in Telco's central office. The main objective here is the tight integration between service management (for placement of VM on servers according to several policies, such as Quality of Service, energy efficiency, locality) and network management (to control connectivity among the large number of distributed installations).

All existing approaches mainly focus on virtualization aspects, but do not consider an overarching middleware that can automatically deploy and orchestrate applications over such distributed environments, including cross-domain scenarios.

[2]Offloading is the process by which a networked device delegates part of its processing tasks, typically to the cloud. Offloading brings several benefits: increased battery life, better response time, more processing power available.

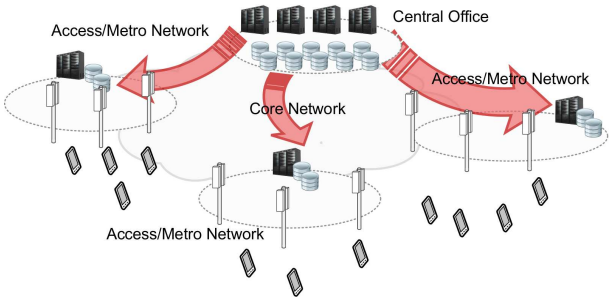


Fig. 1. Edge computing moves computation and storage resources at the network border.

## III. Edge computing architecture

The growing "cloudification" wave and the need to support low-latency applications in a more effective way have motivated the interest in new cloud paradigms, which envision deployment of applications and services at the network edge. Telco infrastructures are typically organized in access, metro, and core segments. Access segments (often referred to as "last-mile") connect customer devices (modems, computers, phones) with the most peripheral network equipment (Broadband Remote Access Servers and similar devices), located in so-called "Points of Presence" (PoPs). Metro networks are used to interconnect several PoPs in densely populated areas (i.e., large cities), while the core network provides wide area connectivity over the entire country served by the operator.

Today, telco networks are rapidly evolving from mere communication infrastructures to rich fabrics of information and communication technologies, with computing and storage resources located close to the border (namely, in PoPs), in addition to traditional installations in central offices (see Fig. 1) [3]. The original purpose was to replace specialized network appliances with software running on plain COTS[3] hardware; however, virtualization technologies also enable to use this peripheral and capillary infrastructure to provide localized cloud services, hence leading to the concept of "edge" computing (recently also indicated as "mist" computing).

The amount of resources in such installations will be limited due to space, power, and cost constraints; in this respect, edge computing is seen more as an extension rather than full cloud replacement [13]. Typical deployments will consist of distributed applications running core processing-intensive tasks in the cloud and lightweight tasks at the edge; the latter will provide functions with strict constraints on latency, bandwidth, quality of service [6].

The European Telecommunications Standards Institute (ETSI) has already published an architecture for Mobile Edge Computing (MEC) [4], which specifically targets cellular

[3]Commercially available Off-The-Shelf refers to general purpose hardware largely available in the market that can be used to implement a wide range of applications and services.

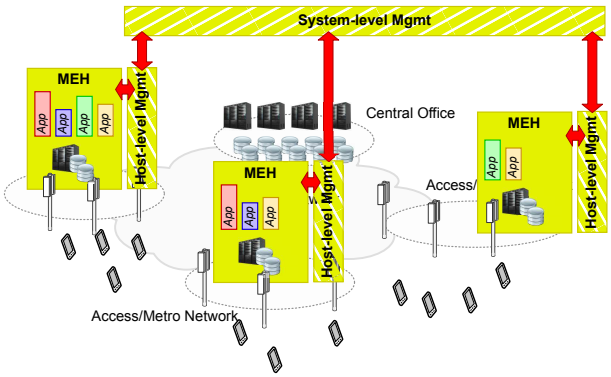


Fig. 2. High-level view of the Mobile Edge Computing architecture.

networks. It is a reference architecture that defines the main architectural elements and their functionality; the objective is an overarching framework for distributed computing, offloading of applications from mobile devices, and onboarding from third parties.

A high-level view of the MEC architecture is shown in Fig. 2. There are two main functional layers: Host and System. The Host layer is present at each Point-of-Presence to run applications, while the System layer provides system-wide management functions.

The Host layer is made of a Mobile Edge Host (MEH) and its management stack (Host-level Management). The Mobile Edge Host is responsible for implementing the execution environment and provides the necessary functions to run user applications (indicated as "App" in Fig. 2) in an isolated sandbox. The execution environment includes both the virtualized infrastructure and ancillary services (for example, to assign IP addresses and automatic configuration at boot, to handle Domain Name System queries, to filter packets, etc.). The management stack controls the Host behavior; this includes provisioning of virtual resources and configuration of ancillary services. System-level Management includes orchestration and placement of applications over the entire set of available Mobile Edge Hosts, and interfaces for managing the whole lifecycle of user applications.

Fig. 3 shows a more detailed view of each architectural block and its internal functions and components [4], [14].

There are two main interfaces to interact with the MEC system at the system level (right side of Fig. 3). The *Customer Facing Service (CFS) portal* is an entry point for providing applications. Software developers use this portal to upload their applications and to make them available to all system users; customers may select the applications they are interested in and specify when and where they are going to use them. The *User Application Lifecycle Management proxy* (User app LCM proxy, in brief) is the function responsible for instantiation, termination, and relocation (to or from external clouds) of the applications. It enables full control of the application lifecycle directly through users' terminals connected to the mobile network. The Operations Support System (OSS) lies in-between the external world and the operator's network. It is widely used in telco's infrastructures to carry out functions like network inventory, service provisioning, network configuration and fault management; in MEC, the OSS performs client authentication and authorization, checks software integrity, relocates the applications between different cloud systems. Requests that are authorized by the OSS are then forwarded to the Orchestrator for further processing. The *Mobile Edge Orchestrator* (MEO) is responsible for onboarding application packages, maintaining a catalog of available applications, and deploying them into Mobile Edge Hosts according to requirements and policies (for example, constraints on latency, bandwidth, resources, services). Further, the Orchestrator monitors the application and trigger lifecycle management actions (such as relocation, scaling, termination).

At the Host layer, the Mobile Edge Host is composed of Virtualization Infrastructure and Mobile Edge Platform, which together run user applications. User applications are software programs previously loaded in the MEC system through the CFS portal. They may run independently, they may use platform services, and they may also provide services to the platform, which can then be consumed by other applications. The *Virtualization Platform* implements the Infrastructure-as-a-Service (IaaS) model and provides virtual compute, storage, and network resources; the data plane consists of virtual networking functions to deliver packets to applications, services, and the access networks. The *Mobile Edge Platform* is a function that discovers, advertises, provides and consumes services, which are provided either by users' applications or by the service provider. The Mobile Edge Platform is also responsible for building a proper networking environment, by inspecting, classifying, firewalling, and steering packets according to deployment policies, and by setting up IP addresses and DNS configuration.

Host-level management entails both Virtual Infrastructure management and Mobile Edge Platform management. The *Virtualization Infrastructure Manager* provides an interface for resource provisioning; it also monitors the usage of virtual resources and reports faults and performance indications to the system-level management entities. The *Mobile Edge Platform Manager* is split into three main functions, which are responsible for *i)* instantiation and termination procedures, including event reporting to the Orchestrator (App. lifecycle mgmt); *ii)* service authorizations, traffic rules, DNS configurations, and conflict resolution (App. rules and req. mgmt); and *iii)* configuration and management of logical network entities, including BRAS[4], DSLAM[5], routers, etc. (Element Mgmt).

## IV. The ARCADIA framework

The ARCADIA framework entails service[6] development, deployment, and orchestration over multiple virtual heterogeneous infrastructures [15]. This includes edge environments,

[4]Broadband Remote Access Servers terminate logical Point-to-Point links in access networks.

[5]Digital Subscriber Lines Access Multiplexers aggregate many last-mile lines into logical flows.

[6]We use indifferently the terms 'application' and 'service' to refer to distributed software.

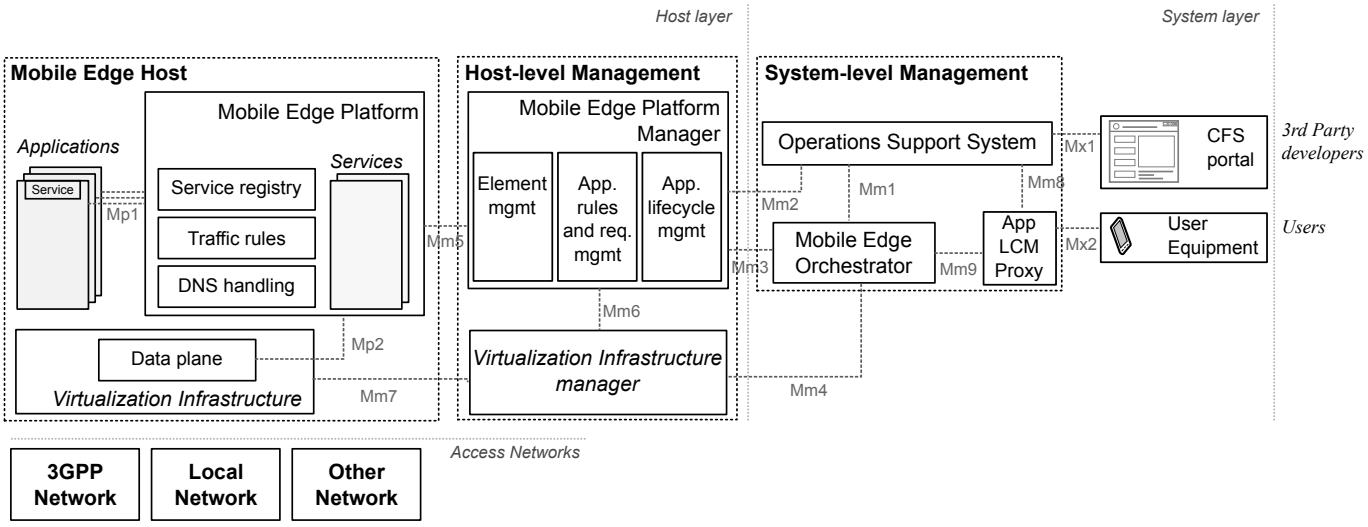


Fig. 3. Mobile Edge Computing reference architecture.

where multiple geographically-distributed sites are used in conjunction with legacy clouds.

The main concept behind the ARCADIA approach is modeling applications in a way to facilitate automatic lifecycle management by intelligent engines. To this aim, ARCADIA applications are designed as "service graphs," which are logical software topologies where nodes represent elementary software units (*micro-services*), and links represent dependencies between them. This structure identifies standalone logical components (i.e., micro-services) that can be run in different locations, and scaled in and out according to the actual workload. The graph includes annotations that set deployment constraints, requirements, and policies since development time. Requirements and constraints can be used to provision the right set of resources to run the application, to select the proper virtualization container and operating system, to install software libraries and dependencies, and to configure micro-services in a consistent way (usernames, passwords, IP addresses). Policies can be used to scale specific micro-services or the entire graph according to the actual workload, to select resources from different administrative domains, to re-configure the system in case of failures, etc. This makes applications elastic and dynamically adaptable to heterogeneous and mutable execution contexts. ARCADIA builds on and extends the TOSCA standard [16].

The overall framework consists of several activities, graphically depicted in Fig. 4.

*Micro-services development* – Each micro-service is an elementary standalone software unit that provides a specific function/service. The executable version of the service is packaged with a control agent and metadata. The control agent can handle any specific lifecycle operation triggered by the system orchestrator. Metadata are organized according to a specific template, said the Context Model, and include: component description, exposed/required functionality, deployment and execution constraints, management policies, monitoring

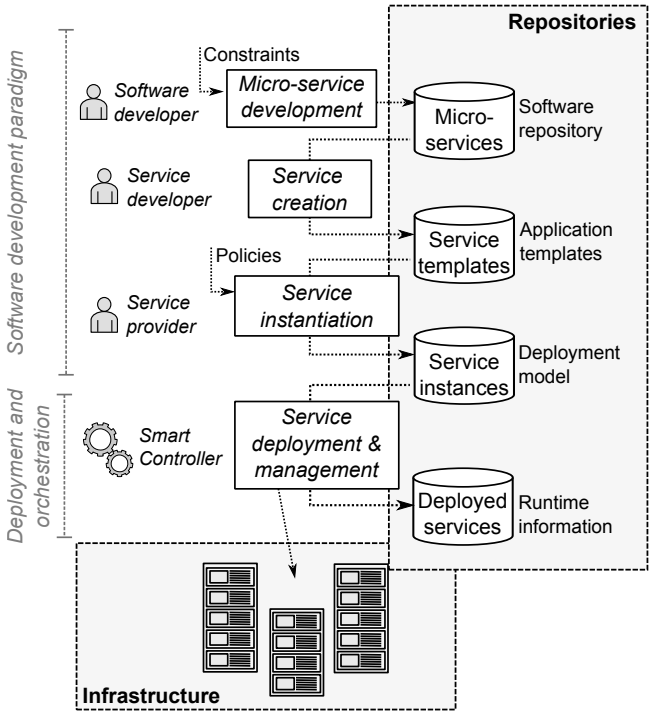


Fig. 4. Conceptual representation of the ARCADIA framework.

hooks, and whatever else may be needed to support automatic deployment and orchestration by smart engines.

*Service creation* – The ARCADIA Context Model envisions the concept of *programming interface*, which includes both exposed/required functionality (for instance: database backend, web server, DNS, etc.) and related configuration options (e.g., username, password, IP address). Such interface is used to create applications, by combining (or *chaining*) micro-services together into graph topologies (*service templates*). Two components are 'linkable' in a graph if they provide

complementary interfaces (e.g., MySQL client and server, HTTP client and server). There is also metadata for the service graph, which contains constraints and monitoring hooks for the whole application, and a programming interface, which makes the application re-usable in other graphs in a recursive fashion.

*Service instantiation* – Service graphs are then enriched with policies to create *application instances*. Policies provide a powerful mean to express optimization objectives, performance thresholds, and actions to manage the entire service lifecycle, from deployment to termination, in an automated way. They account for horizontal and vertical scaling in case the workload changes, resiliency, and migration in case of failure, redundant deployments for backup, and so on.

*Service deployment and management* – Finally, service graphs are automatically deployed by a smart orchestrator, starting from the application package that includes the graph topology, micro-services definition, deployment constraints, optimization objectives and other policies. The smart controller also manages their lifecycle; it undertakes a number of tasks:

- translates metadata and policies to optimal infrastructural configuration;
- computes the optimal placement of micro-services in different execution environments;
- provisions computing, networking and storage resources;
- sets up the execution environment by booting images, installing software and libraries, and running configuration scripts;
- monitors relevant parameters and performance, analyzes measurements and makes predictions about workload, application status, failures;
- shapes the application and its execution environment to the current conditions;
- performs management actions in case of workload changes, failures, and saturation (for example activation, scaling, termination).

## V. The ARCADIA architecture and its usage for edge computing

The overall ARCADIA architecture is depicted in Fig. 5 [15], with explicit reference to the ETSI MEC framework. It basically consists of three main components: the interactive Development Toolkit, an intelligent orchestrator called Smart Controller, and the Execution environment.

### A. Development Toolkit

The ARCADIA Development Toolkit enables development of micro-services and service graphs through an intuitive and immediate Web User Interface (Web UI), which also includes plugins for Integrated Development Environments (e.g., Eclipse Che[7]). Developers implement software as ARCADIA micro-services, including metadata for their correct instantiation and execution, and upload them into the micro-services repository; telco operators as well are expected to provide network-related micro-services, for example functions related to network status, mobility and QoS management. Hence, the Development Toolkit represents a superset of the functionality of the MEC CFS portal. Note that the micro-services repository at this layer is mainly conceived to store their description (i.e., metadata), which is necessary to create service graphs; software images are stored in each execution environment by the Smart Controller.

Applications are created through the Graph Composer, by dragging available micro-services and dropping/chaining them; they are automatically saved as *service templates*. Applications may be developed by the same users (as foreseen in MEC) or by third party developers. When a user wants to activate an application, he picks the service graph from a list and annotates deployment policies and constraints (locality and mobility, QoS, security and privacy levels, response latency, redundancy and resilience level, lifecycle management, etc.), hence creating a *service instance*. The Web UI permits full control on the application lifecycle (start, stop, re-start), therefore playing the same role of the App LCM Proxy in the MEC architecture.

### B. Smart Controller

The Smart Controller splits orchestration into different functional modules. A *Deployment Manager* takes the application model instance and translates it into an optimal deployment plan, by solving a placement problem that takes into consideration available infrastructures (Mobile Edge Hosts, external clouds), their current load and performance, deployment policies and constraints. The plan includes: *i)* the placement of each micro-service to a specific installation, *ii)* additional micro-services and functions to be deployed (for instance, to create virtual networking across different cloud providers), *iii)* the mapping of each micro-service to its execution environment (VM or software container), and *iv)* the list of deployment actions for each micro-service and the entire graph (Operating System, libraries, configurations, network topologies, etc.). The Deployment Manager also keeps an updated view of the whole system, based on deployed Mobile Edge Hosts, available resources, running applications and micro-services, and topologies. An *Optimization Engine* continuously processes monitoring information from host-level components about micro-services and service graphs, and triggers re-configurations as needed (re-placement, vertical/horizontal scaling, and any other lifecycle management action set by accompanying policies), pursuing close-to-zero service disruption. The *Resource Manager* keeps an inventory of all available infrastructures: Mobile Edge Hosts, internal and external clouds. It maintains identity information and authentication secrets and translates internal high-level calls into platform-specific APIs to the Virtualization Infrastructure Management Software for provisioning resources, configuring the execution environment, starting and stopping applications. The combination of these components implements all the orchestration functions envisaged by the ETSI framework.

[7]https://eclipse.org/che/.

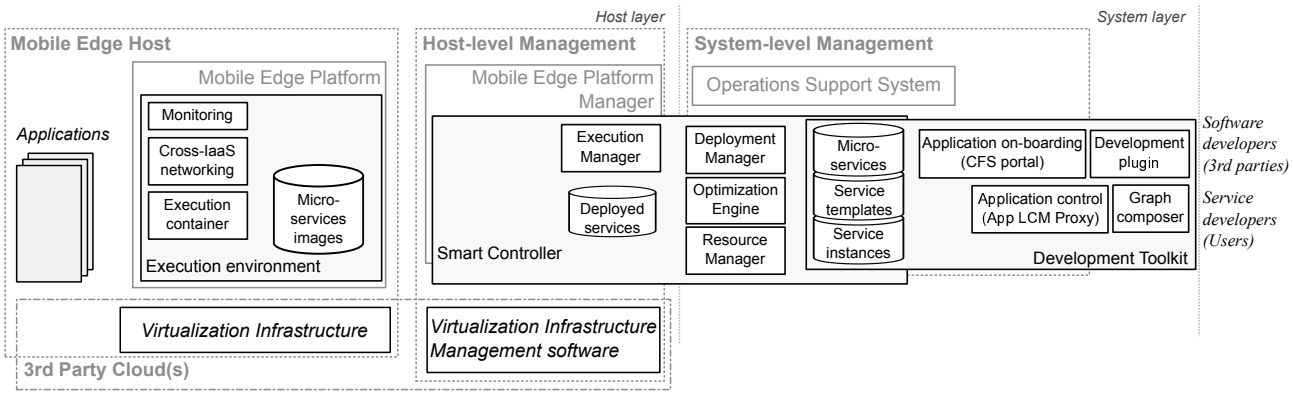


Fig. 5. ARCADIA architecture and its mapping to ETSI MEC for mobile edge computing.

The *Execution Manager* carries out deployment actions, including provisioning and configuration of the execution environment, retrieval of missing software images, micro-services configuration, execution of lifecycle hooks; it also collects monitoring measurements from both the infrastructure management software and specific probes available in each micro-service. The Execution Manager is also responsible for setting up virtual networking, configuring access policies and packet filtering. Run-time information is kept about active instances, their configuration, their status, and any other relevant information for lifecycle management; it can be used by the orchestrator to dynamically connect services, whenever the same micro-service (like databases) could be shared among different applications. Run-time information plays a similar role of the "Service registry" in the MEC framework. All together, the Execution Manager and the run-time repository implement the Mobile Edge Platform Manager in the MEC architecture.

The specific software used to manage the virtualized infrastructure is not relevant for ARCADIA, since the framework is conceived to support cross-IaaS domains. The design of the Resource Manager allows to simply add drivers for different management APIs: OpenStack, Amazon Web Services (AWS), Open Cloud Computing Interface (OCCI), etc.

### C. Execution environment

The Execution environment is deployed by the Smart Controller when the infrastructure is registered in the Resource Manager. Its main purpose is to provide common functions to all software deployed in the IaaS. For instance, it sets up virtual secure networking among all managed infrastructures. The execution environment also collects monitoring information and delivers it to the Smart Controller. This barely corresponds to the functions of the Mobile Edge Platform for MEC.

## VI. Implementation and Evaluation

### A. Implementation

The ARCADIA framework is mostly written in Java, by using the Spring Boot framework[8]. The ARCADIA Dashboard

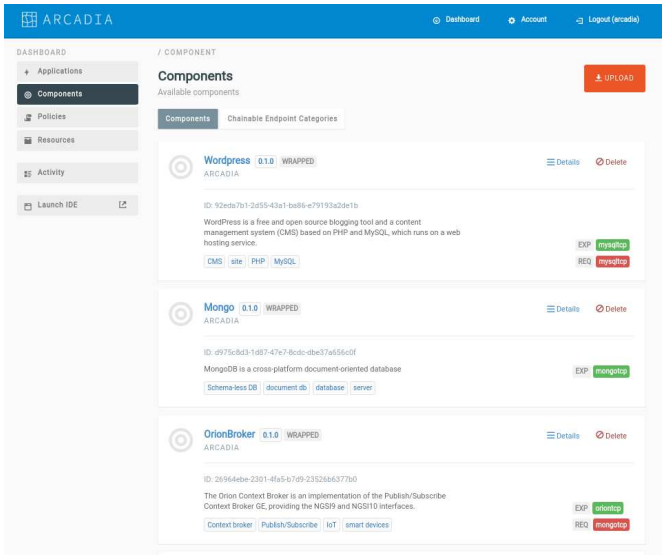


Fig. 6. Screenshot of the ARCADIA Dashboard. The 'Component' menu shows available micro-services, including a brief summary of their characteristics (name, type, version, description, exposed/required interfaces, etc.)

is the entry point that gives access to development tools and the Smart Controller (see Fig. 6); the graphical user interface is built with ThymeLeaf library[9]. Through the Dashboard, it is possible to upload, edit, list and manage micro-services, service graphs, policies, infrastructures.

Micro-services can be developed from scratch as *native* components or ported from existing software as *wrapped* components. The big difference lies in the architecture of their control agent. Both components are controlled by a thin-layer (agent), which is responsible for interacting with the orchestrator through a universal API. However, in the case of a native component, the agent and the component run in the same execution context (the agent is automatically injected in the class-loading process of the component main method), while this is not possible for a wrapped component (software binaries are used, so the agent logic and the handling of

[8]https://spring.io.

[9]http://www.thymeleaf.org.

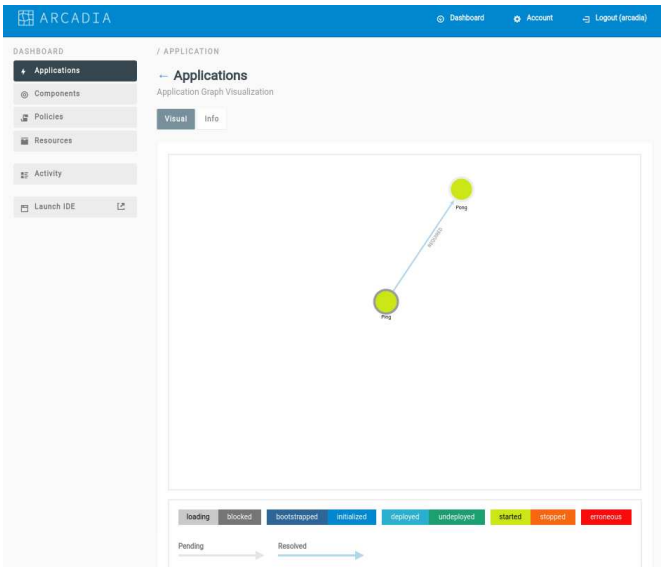


Fig. 7. Screenshot of the ARCADIA Dashboard. The 'Applications' menu allows editing of service graphs, as well as control of their lifecycle (colored boxes at bottom).

the agent signalling should be separated). To solve this kind of issue, the delivery of any wrapped component should be performed using container images; the Open Container Image specification[10] is currently used to resolve any issue regarding binary diversification. Native components, which do not suffer the same limitations, may also be bundled in the form of VMs or unikernels.

The Dashboard enables simple and intuitive composition of micro-services into full applications (see Fig. 7); the user interface prevents logical errors during the construction of the graph. The output of the graph editor is a valid XML representation that can be inserted in the repository. The software repositories build on database technologies, like MongoDB, neo4j, MySQL.

Deployment of a service graph implies the traversal of the directed acyclic graph to resolve first the components that have zero dependencies. After the instantiation of the 'leaves,' the graph is traversed hierarchically to the top (through a recursion) until the root component is deployed. The deployment also entails a placement problem, which can be viewed as combination of two known theoretical problems, the MultiResource Generalized Quadratic Assignment Problem (MRGQAP) and the Unsplitable Flow Problem (UFP) both proved to be in the category of NP-Hard problems. Thus, exact solution is appropriate for small instances of the problem while efficient heuristics are required to find near optimal solutions in reasonable time. OptaPlanner[11], an open source constraint satisfaction solver, is employed to implement the optimization engine. It combines sophisticated optimization heuristics and metaheuristics (such as Tabu Search, Simulated Annealing, and Late Acceptance) with very efficient score calculation to find near-optimal solutions to NP-hard problems in short times.

Internally, policies are expressed by Drools rules-based management system[12]. Each policy consists of a set of rules; each rule is made of a condition and relative actions. Actions include lifecycle management (e.g. start, stop, destroy), runtime configurations (e.g., number of multiple incoming connections), horizontal and vertical scaling, IaaS management. Policies and rules are edited by the Dashboard and associated to service graphs.

Conditions in policies heavily rely on the availability of runtime execution information. Monitoring is supported through a set of active and passive probes, which collect information regarding availability and usage of physical resources (compute, storage and network), resource usage per deployed micro-service, and custom metrics on micro-services and service graphs defined by software developers. Monitoring uses the ActiveMQ Pub/Sub framework and signaling protocol developed in Java; measurements are currently processed by a simple analysis and by a prediction engine implemented in Java, but the Kafka framework is going to be used for this purpose in next releases.

For resource management in multi-IaaS domains, the cloud abstraction library makes use of openstack4j[13]. Resource management registers the whole set of infrastructure providers and provides a framework to setup the execution environment. The execution environment consists of a Linux cloud image that is used to implement the following functions: *i)* running wrapped component bundling; *ii)* build the overlay networking; *iii)* reporting measurements. Virtual networking is realized by Open Overlay Router[14] (OOR), which creates an overlay of secure connections. OOR steers traffic among the micro-services, both within the same and different virtualization infrastructures; it relies on the LISP (Location/ID Separation Protocol) for the control plane and NetConf/Yang for the management plane.

### B. Scalability

Targeting large multi-IaaS environments, scalability is of paramount importance for the ARCADIA framework. The main design aspects that account for scalability are the following:

- support for a huge number of micro-services and service graphs, by using proven database technologies to build the necessary software repositories;
- ability to deploy and manage a large number of applications, by using heuristic techniques to solve the constrained placement problem;
- usage of containerization for reducing the virtualization overhead;
- virtual networking among a large number of IaaS, by using the LISP protocol, which was originally designed to support IP mobility over the whole Internet.

[10]https://github.com/opencontainers/image-spec
[11]https://www.optaplanner.org.
[12]https://www.drools.org.
[13]http://www.openstack4j.com.
[14]http://www.openoverlayrouter.org/.

### C. Preliminary performance analysis

To evaluate the proposed framework, we implemented a simple encrypted Voice-over-IP communication service (VoIP). This application is composed of five micro-services, that realize the necessary functions to manage signaling, VoIP data, registration and authentication, and traversal of NAT devices (Network Address Translation). The objective is to assess the time to automatically deploy the service in a distributed environment.

Preliminary performance analysis has showed that the framework can effectively reduce the time to develop and deploy software in multiple IaaS environments. The ability to compose application through the GUI reduces the development times from days to minutes. It has also been estimated that software maintenance time should be reduced by 30% with respect to current practice.

Real deployments of micro-services took 30 seconds instead of 60 seconds, while the deployment of whole simple graphs required less than 2 minutes instead of 10 minutes. Solving the placement problem took less than 30 seconds, and scaling a service graph required approximately 10 seconds; baseline values with other approaches are about the double of these results. Though these figures are rather indicative, as the deployment time depends on the graph complexity, they demonstrate the big potential behind the proposed approach and implementation.

## VII. Conclusions

In this paper, we have outlined an effective middleware for edge environments, based on the ARCADIA framework. In our opinion, the proposed solution enhances the current architecture for mobile edge computing, by providing a flexible and integrated way of developing and deploying distributed applications.

We believe that further enhancements are still required at infrastructure level to better cope with the peculiarity of mobile environments. First, better mobility support must be available, to allow applications to follow users and their devices while they move around the system. As a matter of fact, OpenStack supports live migration of VMs between servers, but cross-domain migrations are still an open research issue; more investigation is needed to hand over software between different hypervisors and to effectively move data between different domains. Second, tighter control of the network should be in place beyond mere virtual overlays, in order to classify and steer data packets, so to take advantage of the increasing level of programmability of telecommunication infrastructures. To tackle such kind of challenges, the INPUT project[15] is developing OpenVolcano [10], a software platform specifically designed for edge computing environments, which exploits in-network programmability and provides better support for mobility and network management.

[15]INPUT: In-Network Programmability for next-generation personal cloUd service supporT, EC H2020 programme, web site: http://www.input-project.eu.


## Acknowledgment

This work was supported in part by the European Commission under the projects ARCADIA (grant no. 645372) and INPUT (grant no. 644672). The work reflects the only view of the authors, and the Commission is not liable for any use which may be made of the information contained therein.


## References


[1] OpenFog Consortium Architecture Working Group, "Openfog reference architecture for fog computing," Technical paper, February 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12)*, Helsinki, Finland, Aug. 17, 2012, pp. 13–16.

[3] D. Soldani and A. Manzalini, "On the 5G operating system for a true digital society," *IEEE Vehicular Technology Magazine*, vol. 10, no. 1, pp. 32–42, March 2015.

[4] "Mobile Edge Computing (MEC); framework and reference architecture," ETSI GS MEC 003, March 2016, v1.1.1. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf

[5] "Edge computing (MEC); service scenarios," ETSI GS MEC 004, November 2015, v1.1.1. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf

[6] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 63–70, March 2015.

[7] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 80–88, March 2015.

[8] Nokia Networks, "Technology vision 2020 reducing network latency to milliseconds," Whitepaper.

[9] L. Velasco, L. M. Contreras, G. Ferraris, A. Stavdas, F. Cugini, M. Wiegand, and J. P. Fernández-Palacios, "A service-oriented hybrid access network and clouds architecture," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 159–165, April 2015.

[10] R. Bruschi, P. Lago, C. Lombardo, and S. Mangialardi, "OpenVolcano: An open-source software platform for fog computing," in *Proceedings of the $28^{th}$ International Teletraffic Congress (ITC 28)*, Wuerzburg, Germany, Sep. 12–16, 2016.

[11] P. Suthar and M. Stolic, "Carrier grade telco-cloud," in *IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, Bandung, Indonesia, Aug. 27–29, 2015, pp. 101–107.

[12] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, February 2015.

[13] K. Bierzynski, A. Escobar, and M. Eberl, "Cloud, fog and edge: Cooperation for the future?" in *Proceedings of the 2nd International Conference on Fog and Mobile Edge Computing (FMEC 2017)*, Valencia, Spain, May, 8-11, 2017.

[14] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of MEC in the Internet of Things," *IEEE Consumer Electronics Magazine*, 2016.

[15] P. Gouvas, C. Vassilakis, E. Fotopoulou, and A. Zafeiropoulos, "A novel reconfigurable-by-design highly distributed applications development paradigm over programmable infrastructure," in *Proceedings of the $28^{th}$ International Teletraffic Congress (ITC 28)*, Wuerzburg, Germany, Sep. 12-16, 2016.

[16] "Topology and orchestration specification for cloud applications," OASIS Standard, November 2013, version 1.0. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf