TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. 2017; **00**:1–23
DOI: 10.1002/ett

RESEARCH ARTICLE

CloudAware: Empowering Context-Aware Self-Adaptation for Mobile Applications

G. Orsini, D. Bade, W. Lamersdorf[1]

Distributed Systems Group, Department of Computer Science, University of Hamburg, Germany

ABSTRACT

Mobile devices are already woven into our everyday life and we became accustomed that mobile apps assist us in a multitude of daily activities and with the rise of the Internet of Things new opportunities to further automatize tedious tasks open up. New functional and user experience requirements demand for further resources and new ways to acquire these, because mobile devices remain comparatively limited in terms of, e.g., computation, storage, and battery life. To face these challenges, current approaches augment mobile applications either with cloud resources (Mobile Cloud Computing, MCC) or with resources near the mobile device at the logical edge of the network (Mobile Edge Computing, MEC) onto which tasks can be offloaded during runtime. But this does not automatically solve the conflict between resource demands and a good user experience, as current solutions prove. It is the dynamically changing context that makes for good or bad offloading strategies. In this paper, we corroborate this finding by first evaluating 40 existing solutions based on a requirements catalogue derived from several application scenarios as well as the ISO/IEC criteria for software quality. Afterwards, we present CloudAware, a mobile MCC/MEC middleware that supports automated context-aware self-adaptation techniques that ease the development of elastic, scalable and context-adaptive mobile applications. Moreover, we present a qualitative evaluation of our concepts and quantitatively evaluate different offloading scenarios using real usage data to prove that mobile applications indeed benefit from context-aware self-adaptation techniques. And finally, we conclude with a discussion of open challenges.

[1]**Correspondence**
Distributed Systems Group, Department of Computer Science, University of Hamburg, Germany

1. INTRODUCTION

Mobile devices like smartphones, tablets and wearables are continuously replacing stationary devices. Along with this trend the porting of resource-hungry desktop applications and mobile-first applications like augmented reality games has led to a constant demand for higher performance and capabilities. However, in addition to limited interaction capabilities, mobile devices lack computational power, storage capacity, energy and they suffer from a network interface with low bandwidth, high latency and intermittent connectivity. And this not only holds for smartphones, but in the near future for a multitude of other (smart) mobile devices that we will encounter in our everyday life with the rise of the Internet of Things. To overcome these obstacles and to allow even more sophisticated applications being used by mobile users, external resources have to be woven into the local execution of mobile applications [1].

But still, the limited bandwidth and the high latency can have a significant impact on the usability and the user experience. Hence, to ensure an acceptable performance when augmenting mobile devices with external resources, it is not always reasonable to rely on centralized cloud resources as the single backend. *Fog computing* [2], a term coined by Cisco Systems and also known as *(mobile) edge computing* [3], *mist computing* [4] or under the concept of *cloudlets* [5], rely on the assumption that it is impractical or even impossible to always send all data across the whole Internet from the mobile devices to the cloud service provider. Accordingly, mobile edge computing aims at providing resources like computational power and storage at the logical edge of the network and through a more geographically distributed platform rather than at centralized spots, as it happens nowadays through e.g. Amazon AWS and Microsoft Azure cloud datacenters.

However, such and similar concepts are posing even bigger challenges also to the mobile applications' capability to dynamically adapt to the constantly changing execution environment. Just to name a few: Information about the current bandwidth to specific resources and a forecast about their future availability, the current battery level or the probability that an offloaded task will be successfully executed are all part of the applications' and devices' so-called context that needs to be integrated into

Prepared using ettauth.cls [Version: 2012/06/19 v2.10]

the distribution strategy. The mentioned concept has led to an increasing demand for software that is able to exploit the potentials of spontaneous interaction and therefore needs to be able to dynamically adapt to the quickly and constantly changing context of mobile ad-hoc scenarios.

But up to now, existing solutions are proprietary or not well standardized as well as highly domain-specific and therefore it is hardly possible to easily generalize the underlying mechanisms [6]. Moreover, these applications are everything but straight-forward to develop, because of a lack of proper tools (e.g. debugging, testing) to support the development process. Early works originating from the domain of MCC [7, 8, 9, 10] address these problems and try to generalize the mechanisms and ease their use in order to allow more mobile applications to participate in resource augmentation. However, most of these either lack support for proper context adaptation and are hence not able to meet the often inevitable latency requirements, which prevents their direct applicability for MEC scenarios.

Filling this gap, this paper, which is an extended version of previous work [11], discusses the potential of self-adaptation in the light of the particular requirements of mobile edge computing, the respective fulfilment by existing solutions as well as the remaining challenges. In previous work [12] the idea of a context-adaptive MEC solution has been presented, which is now extended to a self-adaptive mobile middleware named *CloudAware* that aims at linking existing concepts of mobile middleware with the specific requirements of MCC and MEC. To provide dynamic adaptation through a configuration-free programming model CloudAware employs compositional adaptation and sensor-based reasoning to allow a flexible adaptation to current as well as future context states that can hardly be foreseen by developers. In particular, we use connectivity- and execution predictions that support the efficiency of the so-called offloading decision in MCC- as well as MEC scenarios. In this way, more generic and flexible scenarios that go beyond just offloading computations become possible. To realize such scenarios and to support a broad range of mobile applications, CloudAware only relies on the presence of a Java Virtual Machine which enables our prototype to augment Android applications without modifying the underlying mobile operating system.

The contributions in this paper can be summarized as follows:

- Categorisation and survey of more than 40 existing solutions and their applicability in MCC/MEC scenarios based on an extensive requirements catalogue derived from application scenarios and the ISO/IEC 25010 standard.
- Presentation of our own context-adaptive CloudAware middleware for the development of MCC- and MEC applications that provides programming abstractions and distribution transparency features without modifying the underlying mobile operating system.
- Qualitative as well as quantitative evaluation of the presented middleware with respect to the derived requirements and based on realistic application and device usage data provided by the *Nokia Mobile Data Challenge* campaign.
- Discussion of open challenges and future work.

The remainder of this paper is structured as follows: Section 2 introduces the foundations of MEC and self-adaptation. Afterwards, Section 3 describes typical application scenarios, that are subsequently used to derive an extensive requirements catalogue in Section 4. Based on this, Section 5 surveys more than 40 existing solutions. Our own approach, the CloudAware middleware is presented subsequently and evaluated in Section 6 and open challenges are discussed in Section 7. At the end, we summarize our findings and give prospects for future work in Section 8.

2. BACKGROUND

In this section we will describe MEC and mobile middlewares as enablers for the greater idea of pervasive computing. Hereupon we will detail a typical MEC architecture as well as granularity levels for offloading. Finally, we introduce the foundations of context adaptation, a feature which enables more sophisticated offloading as used in our CloudAware middleware.

2.1. Mobile Edge & Pervasive Computing

While Mobile Cloud Computing tries to push the limits of mobile applications by including centralized resources to e.g. perform computational offloading, Mobile Edge Computing goes further by assigning the major part of remote operations directly to the surrounding infrastructure - an approach that has been successfully applied to improve latencies of edge-cloud-applications and mitigates the ever growing bandwidth requirements [13, 14]. Here, the resources, located at the logical edges of a network, can include LTE base stations as well as routers providing shared resources [15] and are often directly connected to the mobile device, as shown in Figure 1. In this context, the concept of Cloudlets has been introduced in [5] to represent the capabilities of such mid-sized computing units, located at the edge of the core network. Deploying replicated parts of a mobile application's business logic onto such edge computers can then bring a large benefit to latency-sensitive tasks like streaming or cloud gaming. Hence we consider MEC as a main enabler for the more generic concept of Pervasive Computing, described next.

Pervasive and Ubiquitous Computing aim at integrating computing capabilities into our everyday life. In this course, smart objects sense their environment and communicate and cooperate with each other in order to adapt to their surrounding's needs [16]. The seamless

Trans. Emerging Tel. Tech. 2017; **00**:1–23 © 2017 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

integration of such (generally) resource-poor and possibly mobile devices, which are in most cases located at the logical edge of a network, requires the consideration of a multitude of different and heterogeneous devices. How a standardized execution environment can help to accomplish this task, will be described in the following.

2.2. Role of Mobile Middleware

Due to the challenges of mobile computing in general, the dynamics of logical and physical environments, the possible heterogeneity of context therein, and the complex interplay of devices in pervasive computing scenarios some kind of abstraction that provides a standardized execution environment, referred to as a (mobile) middleware, is required[1]. While traditional middleware has been designed to glue together different pieces of software by providing a set of high-level exchange mechanisms, the purpose of mobile middleware goes beyond. Traditional middleware for stationary distributed systems has been designed with the presumption of high bandwidth, stable connections and high resource availability in mind. Such middleware cannot easily be deployed in a mobile scenario that suffers from effects like intermittent connectivity and requires an asynchronous connection as devices are constantly entering and leaving the ad-hoc established distributed system. Therefore, mobile middleware requires to constantly monitor the users', devices' and environment's context. But as the changes in the context can be highly dynamic they cannot be completely foreseen by the developer, requiring the mobile middleware to anticipate the ever changing context and dynamically reconfigure execution parameters at runtime - namely to perform dynamic adaptation. But there is another issue compared to traditional middleware that is designed to hide the low-level infrastructure details to provide a high level of transparency to the applications running on top. It has been found useful for mobile middleware to provide some extent of awareness to the applications to allow reconfiguration techniques to gather information about the actual execution context [17]. How this reconfiguration in terms of computation offloading can be achieved and what building blocks within a middleware for mobile edge computing are required, will be detailed in the following.

[1] In the remainder of this paper, we will refer to all of the mentioned concepts by MEC only.

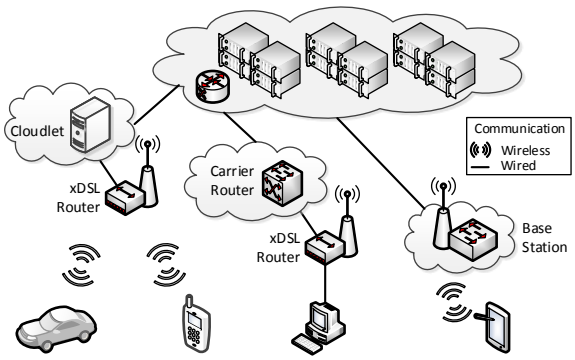

Figure 1. MEC architecture and use-case [12]

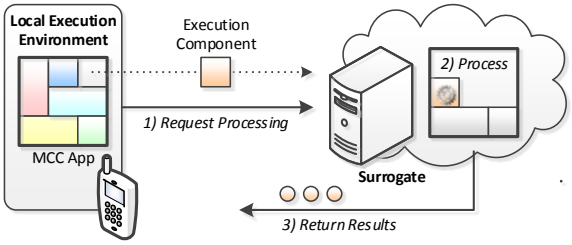

Figure 2. Offloading Components to Surrogates [18]

2.3. Typical MEC Architecture for Offloading

Figure 2 depicts a typical reconfiguration or offloading process respectively. A mobile application, built up from several components, is executed on a mobile device. As one of the components encapsulates a computational intensive task it is moved together with all required state information to a so-called *surrogate* for further processing. This surrogate does as requested and finally returns (only) the results back to the original application.

To enable such kind of computation offloading, existing MEC solutions typically employ some common components [19]:

- A *partitioner* that analyzes the application and determines which parts of the code are offloading candidates.
- A *context monitor* that senses contextual information like available surrogates, battery status and network connectivity.
- A *solver* that uses information from the partitioner and the context monitor to decide, whether and on which surrogate to execute the offloading candidates.
- A *coordinator* that handles additional necessary tasks like discovery, authentication and synchronization.

One of the main challenges in MEC is making an optimal offloading decision. Based on [20] this includes the following: Which components are potential offloading candidates? Under what circumstances shall a component be offloaded and how to perform? And where can the component be offloaded to? The first question aims at identifying which components may benefit (in terms of execution time, for example) from being offloaded as well as finding an appropriate offloading granularity for a good partitioning of an application and its constituent components. Answers to the second question need to take the current context of the user, the device, the execution

platform and the application's execution state as well as of the logical and possibly physical environment into account. This is due to the fact that creating an optimal deployment strategy, i.e. successfully and efficiently offloading a component and receiving the results back, is strongly dependent on the current context as will be detailed later on. Finally, to answer the last question, where to offload a component to, one has to consider characteristics of possibly targeted offloading nodes in terms of available resources and their workload, the quality and continuity of the connection and the security of this node.

2.4. Offloading Granularity

When solving the aforementioned offloading decision, different offloading granularities are possible. While there exist many other granularities, many solutions can be broadly classified into the following categories[21]:

Method offloading uses what is commonly known as remote method invocation to perform computation offloading and is often used in scenarios, where fine-grained control is necessary. Instead, offloading a (self-contained) feature such as, e.g., face recognition, is best reflected by the classical client-server paradigm where a client requests the execution of a parameterized remote operation and subsequently receives the result back from the server. These two ways of offloading are already widely used and integrate very well with the principles of object-oriented design, but the constant need for synchronization of shared variables between client and server is the main drawback [8]. Distributed object frameworks like *Jini*[2] often make use of this principle, but they generally lack the ability to adapt invocations to the current context, e.g. in the case of intermittent connectivity.

The remaining two categories can be summarized as virtual machine (VM) approaches. They consist of 'image' or application layer offloading (e.g., *Jikes rVM* used in [22]) and system layer offloading (as used by, e.g., Goyal et. al. [23]). The advantage of application layer VMs is that they abstract from the underlying CPU design (*ARM*[3]-CPUs are often used for mobile devices, while *x86*-CPUs are used on the surrogate side), while system-level virtualization is favorable, if the whole environment of the mobile device needs to be mirrored on the surrogate (e.g., to catch file system calls).

While feature and method offloading typically allow the developer to control the partitioning and offloading of an application into the infrastructure, this is typically not the case for VM-based approaches that are not able to fully catch the developer's expertise and have to rely on heuristics to optimize the partitioning and execution strategy which can be a highly complex task when it comes to scenarios with intermittent connectivity where adaptation to future contexts is required. Another drawback of VM-based approaches is the requirement to synchronize a high amount of state between the mobile device and the surrogate, which often restricts the offloading to one task (meaning method or thread) at a time in order to maintain consistency and thread safety, though this approach allows to offload unmodified applications to the infrastructure.

2.5. Self-Adaptation

We consider an early definition applicable to our perception of self-adaptation as referred by Laddaga in [24]:

> *"Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."*

Current patterns for self-adaptation can be briefly differentiated in two classes [25]: Parametric adaptation and compositional adaptation. While the adaptation rules in the former are often woven into the business logic, compositional adaptation in contrast follows the paradigm of the separation of concerns. Taking into consideration the simple example of three parameters of which each can have 4 states - already 64 different states are possible and need to be considered by the developer. Considering real-life examples that have way more parameters and states, as also found by [26], it becomes clear that compositional adaptation provides a far more flexible concept to develop self-adaptive applications - whose three main enablers, according to [27], are the separation of concerns, computational reflection, and component-based design. A well-known concept in this context is the paradigm of aspect oriented programming, presented in [28]. When the separation of concerns is achieved, computational reflection then allows the program to reason about its own behavior and possibly alter it, if required [27]. Lastly, the component-based design is the paradigm that enables self-adaptation, as it requires the developer to partition an application into self-contained units, whose single building blocks can either be exchanged at buildtime (static composition) or at runtime (dynamic composition).

According to [26], self-adaptive systems can be further differentiated by the level of anticipation they provide. If a developer predefines the systems behavior, the degree of anticipation is lower than in the case where no rules are defined and the anticipation to the current context happens dynamically at runtime, referred to as context awareness. For a detailed definition and a complete survey of the most relevant context modeling approaches, we refer to [29].

Recent MEC solutions consider context adaptation only as a minor factor to allow offloading parts of the computation to a surrogate, while we consider it the essential criterion to enhance the user experience, which will be reflected upon in Section 6.

[2] http://river.apache.org
[3] http://infocenter.arm.com

Trans. Emerging Tel. Tech. 2017; **00**:1–23 © 2017 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

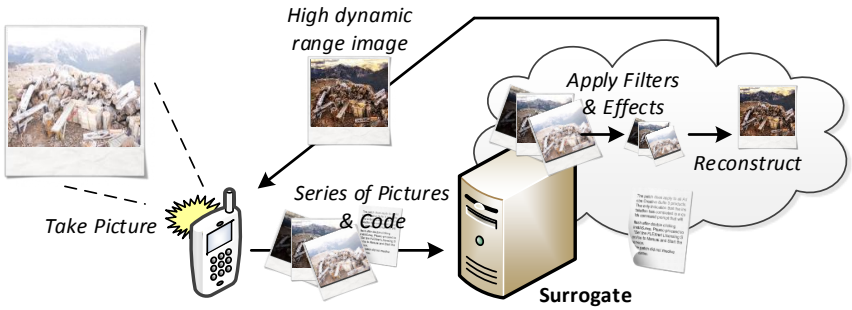

Figure 3. Application Scenario: Image Processing

3. APPLICATION CHARACTERISTICS

To further illustrate the idea of MEC, four application scenarios are exemplified in the following from which the first serves as a base for the evaluation presented in Section 6. Subsequently, the generalized main characteristics of MEC applications are presented.

3.1. Image Processing

As all current smartphones have a built-in camera, mobile image processing gains more attraction. And with the advent of high-resolution cameras (e.g. Nokia's Lumia with 41 Megapixel) respective tasks become even more challenging. Moreover, also professionals, journalists for example, use smartphone cameras to take snapshots, which subsequently undergo typical processing steps like noise reduction, color enhancements, object extraction, masking, etc. before being published. Finally, more complex tasks like image stitching to create 360 degree panorama images or face recognition raise the bar even higher.

Imagine you are on a trip abroad and you took a couple of pictures and need to process them before sending them back to your office or your friends (see Figure 3). Image processing, especially on high resolution images, can be a resource-intensive task, e.g. creating high dynamic range images out of a series of pictures. You do not want to exhaust your mobile's resources and instead offload this task (pictures as well as code) to a surrogate and hence substitute processing with communication.

3.2. Interior Designer

Suppose you would like to rearrange your interior[4]. Using your smartphone's camera, you walk around in your apartment taking a series of pictures in order to digitally reconstruct the scene. Because 3D reconstruction using multiple images is a computationally expensive task [31] the photos as well as the reconstruction code are uploaded into the nearby edge cloud. Here, the scenes are matched, objects as well as the room geometry are photo-optically surveyed and a complete 3-dimensional model of your apartment including models of your furniture is constructed. The results are subsequently sent back to the smartphone. Looking through the live video stream of your camera (or your smart glasses) you are now able to virtually rearrange single pieces of furniture, detect collisions, and see how the lighting is influenced. This time, the smartphone only needs to extract image features based on your interaction and request the construction of appropriate overlays by the edge cloud. The low-latency requirements of this scenario are viable for a smooth application usage.

3.3. Bitcoin Validation

With the increasing popularity of bitcoins[5], more and more payment transactions are carried out using this new currency[6]. Along with that, the allurement of fraud increases as well. In order to validate a transaction, one uses the so-called 'blockchain', a several gigabyte big and continuously growing file containing a complete history of all transactions ever made within the bitcoin network [32]. Suppose you are on a flea market and shall receive a certain amount of bitcoins for selling some of your belongings. To make sure that the bitcoins are not spent twice and really change ownership, you have to wait for several minutes or even hours[7] until the transaction is validated by so-called miners and appended to the public blockchain. For this purpose you can repeatedly download and check the end of the blockchain. But doing so consumes quite some bandwidth and energy. Alternatively, the task can be delegated onto a nearby edge device by offloading the validation component (along with the transaction details), which continuously checks and validates the blockchain and proactively informs you once your transaction is succeeded.

[4] This scenario is inspired by [30].

[5] http://www.bloomberg.com/news/articles/2016-01-07/the-return-of-bitcoin-mining, accessed 31.10.2016

[6] https://blockchain.info/de/charts/n-transactions-total, accessed 31.10.2016

[7] http://www.ibtimes.com/bitcoins-big-problem-transaction-delays-renew-blockchain-debate-2330143, accessed 31.10.2016

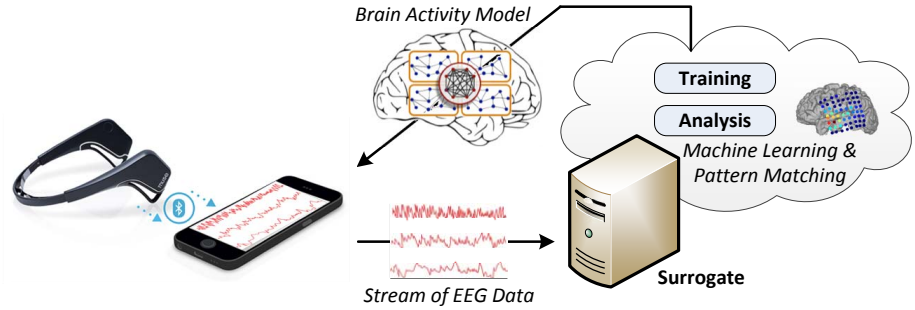

Figure 4. Application Scenario: Brain Wave Analysis

3.4. Brain Wave Analysis

Attention deficit hyperactivity disorder (ADHD) is a mental disorder children are quite often diagnosed with[8]. One new approach to cure the symptoms are brain-controlled computer games, which force the children to concentrate if they want to win the game[9]. For this purpose, children wear a special headband or a kind of bike helmet with an integrated EEG[10], measuring electrical activity along the scalp. Processing and analyzing the signals from the multiple electrodes is computationally expensive and too much of a burden for ordinary mobile devices. But with the advent of low-cost consumer EEGs (already available for less than 100 Euros) this kind of therapy might gain more importance and MEC offers means to deal with the near real-time analysis of brain waves even while on the way.

For this purpose, a model of a child's individual brain wave pattern has to be trained initially in order to distinguish mental states like 'calm' and 'active'[11]. Figure 4 depicts how EEG data is fed into machine learning algorithms to adjust their parameters.

Once the model has been trained it can be stored onto the mobile device. Later on, when the child plays the game, the algorithm as well as the trained model are uploaded onto a surrogate. Then, the live EEG data is forwarded from the mobile device to the surrogate where it is classified according to concentration levels and only the resulting neurofeedback is returned to the device to alter the current state of the game.

3.5. Generalized Application Scenarios

All of the presented application scenarios (and further ones, see [33, 12]) have a least common denominator: The device the user interacts with has resource constraints, but the tasks require either computational power (and energy), loads of data, network bandwidth or additional resources. Depending on the context, the tasks can be offloaded to surrogates like edge cloud devices or a cloud platform where they are processed and the results are finally sent back to the initiating device. In contrast to traditional communication means (e.g., remote method invocation (RMI) or service calls), the logic to process the tasks is sent along with the request, hence, the surrogates do not have to be set up beforehand. This kind of architecture, together with proper context adaptation, can be considered the foundation of successful future MEC applications.

To conclude, the primary objectives for using edge nodes as intermediaries between mobile devices and clouds are:

- Provide services or deliver content close to the user's access point (LTE or WiFi)
- Process or filter large amounts of data before they are transferred
- Speed up geo-distributed applications like sensor networks
- Perform distributed large-scale analysis of real-time data
- Allow latency-sensitive applications like cloud gaming and real-time video analytics

4. MAIN REQUIREMENTS

This section presents an extensive list of requirements, based on the ISO criteria for software quality [34] and inferred from the previously presented use cases. In the following, we present the foundation for our survey and evaluation of existing research efforts in Section 5.

4.1. Major Challenges in MEC

For several reasons, the development of MEC applications performing computation offloading is a difficult task and needs to be supported by appropriate and specialized tools. On the one hand, established mechanisms like inter-process communication and remote method/service invocation cannot be employed in this context [35]. And on the other hand, developers do not only have to

[8] http://www.cdc.gov/ncbddd/adhd/facts.html, accessed 11.04.2016
[9] http://www.insideadhd.org/Article.aspx?id=1386, accessed 11.04.2016
[10] Electroencephalography, monitoring method for brain activity
[11] See, e.g., http://www.choosemuse.com/how-does-muse-work, accessed 11.04.2016

Trans. Emerging Tel. Tech. 2017; **00**:1–23
DOI: 10.1002/ett
Prepared using ettauth.cls

deal with the challenges of classical distributed systems, namely the heterogeneity of their constituent components and the requirements of openness, security, scalability, failure handling, concurrency, transparency and quality of service [36], but are required to additionally deal with the restrictions imposed by MEC itself, being the limited resources, the need for context adaptation in heterogeneous environments as well as security issues.

Over and above these challenges, an MEC solution should also comply with general criteria for software quality as defined by *ISO/IEC 25010* (former ISO 9126) [34], which comprise a software's reliability, usability, efficiency, maintainability and portability. In order to develop a comprehensive list of general requirements for MEC applications, we started bottom-up with adapted ISO requirements and merged them from a top-down perspective with specific requirements inferred from the application scenarios presented in the previous section to ensure the requirements have practical relevance. The resulting list is summarized in the following (further details can be found in [37]).

4.2. Main requirements based on ISO 25010

We conclude that the aforementioned restrictions require an appropriate support by an underlying infrastructure which we describe as an *MEC framework*. Based on this assumption we develop criteria covering the aspects of availability, portability, scalability, usability, maintainability and security as follows:

Availability

Availability of surrogates has a direct impact on the successful execution of offloaded tasks. It can be further subdivided into four different aspects:

A1 Error Handling: *Connectivity failures and transmission errors between mobile devices and surrogates should be intercepted and a local execution of the current task initiated.*

A2 Forward Error Management: *Intermittent connectivity issues and excessive load on surrogates should be recognized and rebalanced by an underlying layer and not interfere with the applications' functional control flow.*

A3 Connectivity Prediction: *Future connection quality and connection status to surrogates should be anticipated and considered for the offloading decision.*

A4 Effectiveness: *Lost connections to surrogates should not lead to blocked resources, dangling pointers or further negative side-effects.*

Portability

Portability requirements are concerned with offloading in general and context adaptation in particular. Fulfilment of such requirements is essential for modern MEC solutions.

P1 General Offloading Capability: *The ability to execute appropriate parts of the code on surrogates and receive the result to shift load from the mobile device to the surrogates.*

P2 Basic Adaptation: *Changes in the context should be considered to adapt to different environments (e.g., to re-evaluate the current offloading strategy).*

P3 Advanced Adaptation: *To allow more sophisticated offloading scenarios, past, current and future (A3) context situations need to be considered in the offloading decision and the MEC architecture needs to be able to support these quickly changing environments automatically (e.g., synchronizing relevant state information).*

P4 Coexistence: *The general concept of fair resource sharing on mobile devices needs to be extended to edge cloud-augmented apps. MEC applications on surrogates should coexist with other applications, hence, not making exclusive use of resources.*

P5 Deployment: *The deployment process of MEC applications should be as easy as installing regular applications, which requires ad-hoc deployment to surrogates for advanced adaptation scenarios (P3).*

P6 Openness: *The framework should be able to interact with different edge cloud service providers by using open standards and interfaces.*

Scalability

In order to increase the user experience and at the same time not to burden the resources of a mobile device, several scalability issues need to be taken into account:

S1 Overhead: *MEC applications, if executed locally, should have similar performance as conventional applications. The framework-based overhead should be minimal, both on the mobile device and the surrogate.*

S2 Discovery & Integration: *The framework should be able to search and integrate new remote resources quickly and rebalance the offloading strategy to enable the opportunistic use of resources and to make use of specialized surrogates (e.g., providing sensors or FPGAs).*

S3 Parallelization: *Parallel execution by splitting up the computation task should be supported, either automatically or by established mechanisms (e.g., threads).*

S4 Efficiency: *Hibernation capabilities for paused surrogate sessions are required to handle numerous connections to a single surrogate and quick re-establishment of these to fulfil the low-latency requirements in MEC.*

Usability

Developers and end-users shall ultimately benefit from adopting an MEC solution and not be forced into a new mindset. Therefore, several usability requirements shall ideally be met.

U1 Ease of Use: *The design and implementation of MEC applications should be easy and intuitive. Domain-specific knowledge should not be necessary and limited to general aspects of distributed systems.*

U2 Appropriate Abstraction: *Distribution transparency, as an appropriate level of abstraction, should be provided by the framework. For example, communication details should be hidden.*

U3 Tool Support: *The development of MEC applications should integrate with the common development tools and the present build processes.*

U4 Hands-free for End-Users: *Configuration requirements on the end-user side should be limited to optimization on overall targets like battery, speed or bandwidth.*

Maintainability

As middlewares and applications evolve, developers need to make sure their software remains maintainable. In this course, following aspects need to be considered:

M1 Determinism: *In all situations the control flow of an MEC application must be deterministic to the user, independent of the used surrogates; just expectable time-lags are acceptable.*

M2 Debugging: *Debugging and maintenance should not be more complex as with a conventional application. Control flow and state should be easily analyzable with common development tools.*

M3 Open Standards: *Use of open standards for communication (e.g., protocols and data formats) should help to ease maintainability, extensibility and should prevent vendor lock-ins.*

Security

Security and privacy are two requirements that must be fulfilled by any solution that is going to be used in practice. This comprises:

SE1 Privacy: *No data should be shared with untrusted devices, unless assigned by the developer or the end-user.*

SE2 Isolation: *Execution of untrusted code should not lead to side-effects on surrogates or compromise other users.*

In conclusion, we identified several requirements that arise from the challenges of distributed systems in general and are extended by MEC in particular which ideally should be met by an MEC framework and against which we evaluated overall 40 existing frameworks, which is presented in the following.

5. EXISTING SOLUTIONS

Throughout the years a multitude of MEC solutions have been published. In this section we first retrace the evolution of MEC approaches. To ease the overview a classification of approaches is presented afterwards. Solutions within the same class share some common characteristics, but mostly have their own special focus on certain aspects. Hence a survey and detailed evaluation follows subsequently.

5.1. Evolution of MEC approaches

In terms of popularity, MEC solutions receive increasing attention. The literature review, carried out with the purpose of finding work that is directly related to the domain of resource augmentation, shows a steady growth over the years, as shown in Figure 5. The earliest approaches like [38] or the Emerald system [39] dates back to 1988. The first peak in 2002 is mainly related to the emergence of VM-based solutions, the more current peak in 2012 primarily originates from solutions that provide a more fine-grained level of offloading granularity. Throughout the years more than 77 different approaches have been published in total. We selected the surveyed solutions primarily by their publicity and for this purpose we counted the cross references of other surveys and publications in the field that we surveyed. Consequently, we tried to balance the six categories accordingly while avoiding to present too many similar solutions.

In the following, we focus our classification and survey on solutions that fit the common mobile application developer mindset and allow to maintain an established toolset, which helps to keep a low entry threshold. Even if other solutions like the MapReduce-based *Hyrax* [40] provide an interesting approach in terms of parallelizing tasks on mobile devices, they inhibit a steep learning curve and are therefore just mentioned for the sake of completeness.

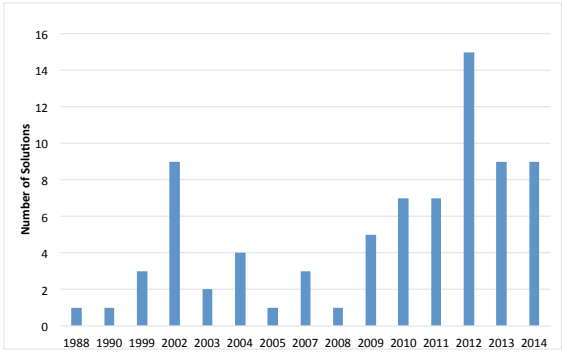

Figure 5. Evolution of Mobile Edge Computing Approaches

5.2. Classification

Surveying existing solutions, we classify these according to their nature (cf. [37]). Consequently, we first classified them by the research strand they originate from and subsequently by the way they are performing the computation offloading. The native MEC solutions are further distinguished by their offloading granularity.

Trans. Emerging Tel. Tech. 2017; **00**:1–23
DOI: 10.1002/ett
Prepared using ettauth.cls

But, as indicated in Figure 6 a clear distinction is hardly possible, because some existing solutions adopt characteristics of more than one class. Next, the six classes will be detailed.

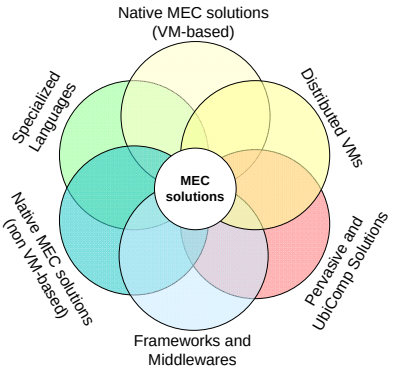

Figure 6. Overview of Classes and their Blurred Bounds

Specialized Languages Directly embedding support for a broad range of heterogeneous offloading scenarios into the programming language itself is incorporated by so-called domain-specific languages. Nevertheless, they require the developer to become acquainted with new language syntax and programming style and require proper support of the mobile operating system, which is currently not the case.

Frameworks & Middlewares Frameworks and middlewares are a well known concept to connect heterogeneous environments and are used to provide a uniform and higher layer of abstraction on top of different underlying systems. Here, the developer is just required to extend the provided infrastructure with application-specific code. Middlewares exist for different programming environments and by using a specific framework, the programmer is bound to that environment.

Distributed VMs In contrast to middlewares that require explicit refactoring of an application, distributed VMs are heading the complete opposite direction. They follow a generic approach to provide a uniform execution layer among different systems that allow the distributed execution of applications without dealing with the distribution details. This high degree of transparency eases the development, but requires adequate heuristics to allow an efficient distributed execution. In terms of intermittent connectivity, these solutions often provide no exception handling, but presume a stable network connection.

Pervasive & UbiComp Solutions Solutions that originate from the domain of smart homes or more commonly ambient intelligence are designed to work on the system level and take care of context data acquisition, resource discovery, data distribution via distributed file systems, automatic partitioning of applications and the distributed execution. They are not explicitly designed for computation offloading, but instead allow to move complete processes between different nodes in a network. Their main strength is to explicitly handle changes in the applications' usage context already in the design phase, by offering a high-level programming interface for the developers. Yet, they require the developer to become familiar with the solution and require proper operating system support for all participating devices.

Native MEC Solutions (non VM-based) Originating from the ever increasing growth of mobile applications, several approaches have been explicitly built for the domain of MEC. They can be broadly classified into those that rely on virtualization approaches and those that require a more explicit handling of the MEC-specific restrictions. For the latter, components depict a common design principle to partition the applications' business logic into self-contained fragments with high cohesion. Nevertheless, most of the current solutions do not provide features for context awareness in order to adapt the execution strategy accordingly. This often leads to suboptimal resource utilization and sometimes prevents the ad-hoc interaction with nearby devices.

Native MEC solutions (VM-based) Next to the aforementioned class there exist several solutions that seek to provide full distribution transparency among the involved devices. Here, complete device images or at least parts thereof are replicated in the infrastructure and allow to offload complete threads or methods.

5.3. Survey and Evaluation

Now that the different classes have been introduced, a survey and evaluation of existing solution will be presented in the following. Table I shows how each of the requirements developed in the previous section is fulfilled by the surveyed solutions. A low score does not generally refer to a bad solution, but just implies that the approach does not match the requirements for context-adaptive MEC applications. Furthermore, one has to consider that some of the solutions were not initially built for MEC scenarios, e.g., the popular *Gaia* approach [52] performs below average in terms of offloading capabilities but superior in the branch of context adaptation, as being initially designed for this purpose. The performance is presented by a distinction whether a criterion is completely met (++), partially met (+), not met (-) or not addressed at all (0). To give an impression on how the scores come about, Table II briefly summarizes the respective strengths and weaknesses of four prominent solutions[12].

[12]For further details we refer to the respective publications as referenced in the table.

Table I. Survey of Existing Mobile Edge Computing Solutions

		A1	A2	A3	A4	P1	P2	P3	P4	P5	P6	S1	S2	S3	S4	U1	U2	U3	U4	M1	M2	M3	SE1	SE2
Specialized Languages	AmbientTalk [41]	-	++	-	++	-	+	-	++	++	0	0	+	++	++	-	++	0	0	+	+	0	0	0
	Kairos [42]	-	-	-	0	-	-	-	-	-	+	-	0	0	0	+	++	0	+	-	++	0	0	0
	Pleiades [43]	-	0	-	-	-	+	-	-	-	-	-	0	0	-	+	+	0	+	-	-	-	0	0
Frameworks & Middlewares	Agilla [44]	-	-	0	0	-	-	-	0	0	0	++	-	-	-	-	-	-	0	0	-	-	0	0
	ASM [45]	0	0	-	0	++	-	-	0	++	-	++	0	0	0	++	++	++	++	++	++	-	-	-
	CoDAMoS [46]	-	-	+	-	+	++	++	++	0	++	0	++	0	0	++	0	++	++	++	-	0	0	0
	Hyrax [40]	+	-	-	++	++	++	+	0	0	++	-	-	++	++	+	+	0	++	0	0	++	++	0
Distri-buted VMs	COMET [47]	++	++	-	-	++	+	-	++	++	++	++	-	++	-	++	++	++	++	+	++	+	-	+
	Jessica2 [48]	0	-	-	++	++	-	-	++	++	++	+	0	++	+	++	++	++	++	++	++	++	0	0
Pervasive & Ubiquitous Computing Solutions	Vivendi/Chroma [49]	++	++	0	0	++	++	0	++	++	++	++	++	++	0	+	++	+	++	++	++	-	-	-
	CADeComp [50]	-	-	-	-	+	-	-	++	++	++	0	0	0	++	-	+	++	0	++	++	++	0	0
	CAwbWeb [51]	-	0	-	-	-	-	-	0	++	++	-	0	-	-	-	+	++	++	0	0	++	-	-
	GAIA/MobileGAIA [52,53]	-	+	-	++	-	+	-	++	++	++	++	++	0	++	++	++	0	++	++	0	-	++	++
	Ghiani [54]	-	-	+	++	-	++	++	++	++	-	-	++	-	0	+	++	++	+	++	+	++	+	++
	MDAgent [55]	-	-	-	-	++	-	-	0	++	++	0	+	0	0	-	++	++	0	++	++	++	++	0
	MOB-Aware [56]	++	-	+	-	++	+	+	0	0	0	++	-	-	-	++	+	++	-	++	++	0	0	0
Native MEC-/MCC-Solutions (non VM-based)	AIOLOS [57]	++	++	0	0	++	++	0	++	++	++	++	0	0	0	++	++	++	0	++	++	++	++	0
	AlfredO [58]	-	-	-	-	++	-	-	++	-	-	++	-	-	-	+	+	++	++	++	++	++	++	0
	CMH Application [59]	0	0	0	0	++	-	-	++	++	++	-	0	0	0	++	++	+	0	0	++	++	0	0
	Cuckoo [10]	++	++	+	++	++	++	+	++	+	++	++	+	++	++	++	++	++	++	++	++	++	+	-
	exCloud [60]	0	++	0	0	++	0	0	++	++	++	++	0	++	0	++	++	++	0	-	-	0	0	0
	Giurgiu et al. [61]	-	-	+	0	++	++	++	++	++	++	++	++	-	++	++	++	++	++	++	++	++	0	0
	Gu et al. [62]	-	-	-	++	++	-	-	++	++	++	++	++	-	+	++	++	++	+	++	++	0	0	0
	Huerta-Can. et al. [63]	-	++	+	++	++	++	+	+	++	+	++	++	-	-	++	++	++	0	++	+	++	++	++
	IC CLOUD [64]	-	0	++	-	++	++	++	0	0	0	++	-	++	-	++	++	++	0	++	++	0	++	0
	MOCHA [65]	-	-	-	-	+	-	-	+	-	++	0	0	+	-	0	0	0	++	0	0	0	0	0
	NAM4J [66]	0	0	-	++	++	+	-	++	++	++	++	++	-	0	++	++	++	-	++	++	++	-	-
	Odessa [67]	++	++	++	0	++	++	++	++	-	-	++	++	++	0	++	++	0	++	++	++	0	0	0
	Scavenger [68]	0	0	-	0	++	++	-	++	++	++	++	++	++	0	-	++	++	++	++	++	++	++	+
	Zhang [69]	0	-	-	0	++	++	-	++	++	++	++	-	++	++	-	-	++	++	++	++	++	-	-
	µCloud [70]	-	-	-	0	++	-	-	++	++	-	0	0	-	0	++	++	++	+	++	++	-	-	-
Native MCC-/MEC-Solutions (VM-based)	AgentISR [71]	++	-	++	-	++	+	-	++	++	0	-	0	++	0	++	++	++	++	++	++	++	++	++
	Chen et al. [72]	-	-	-	+	++	-	-	-	++	++	++	-	0	+	++	++	++	0	++	++	++	-	-
	Cirrus Clouds [73]	-	++	++	0	++	++	++	++	++	++	++	++	++	0	++	++	++	++	+	-	-	0	0
	CloneCloud [8]	0	0	-	0	++	-	-	++	++	++	+	+	-	0	++	++	++	++	+	-	++	-	0
	Jupiter [74]	+	-	-	0	+	0	0	++	-	++	++	0	0	0	++	++	0	++	++	++	++	0	0
	MAUI [7]	++	-	+	++	++	++	+	++	++	++	-	+	-	++	++	++	++	++	++	++	-	++	0
	Slingshot [75]	++	+	-	++	++	-	-	++	++	++	++	+	-	+	+	+	++	-	++	++	++	-	-
	ThinkAir [9]	++	++	+	++	++	++	+	++	+	++	++	+	++	++	+	++	++	++	++	++	++	-	-
	Hung et al. [76]	0	-	-	++	++	++	0	++	++	++	0	0	++	0	-	-	++	++	++	-	0	++	++

5.3.1. Summary of the Evaluation

Summarizing the analysis and evaluation, it can be concluded that the usability and maintainability of the surveyed solutions are performing equally well. We argue that this is the case because most solutions try to integrate into common development tools and build processes. Though, the ease of use (U1) could be enhanced. Another aspect, performing equally well, is the portability of the evaluated approaches. Here, coexistence (P4) and deployment (P5) are receiving top scores, while basic adaptation (P2) and in particular advanced adaptation scenarios (P3) still could experience improvement.

Trans. Emerging Tel. Tech. 2017; **00**:1–23
DOI: 10.1002/ett
Prepared using ettauth.cls

Table II. Excerpt of surveyed MCC solutions

	CloneCloud	ThinkAir	Cuckoo	eXCloud
Availability	no connectivity loss recovery, no forward error management	connection recovery, forward error management, connectivity prediction, dynamic surrogate usage	connection recovery, forward error management, connectivity prediction, temporary surrogate for execution measurement	automatic migration from unavailable or busy surrogates
Portability	not context-aware, offloading strategy chosen at startup, no vendor lock-in	adaptation through connection profiling, provides multiple VM profiles	offloading decided at runtime, surrogate reachability checks	duplicates methods at bytecode level, fine grained offloading
Scalability	simple deployment, bad discovery	automatic scaling through VM-Manager,	surrogates must be preregistered	dynamic migration
Usability	automatic transformation of applications, no code modifications, no additional setup by users	code annotations required, selection between energy- or time-efficiency, provides UI for configuration	adaptations for Android AIDL, selection between energy- or time-efficiency, surrogates must be preregistered	transparent movement of code to a VM-based surrogate
Maintainability	bytecode and threading information for debugging	exceptions rethrown on client, debugging through virtualization	debugging through IDE integration, offloaded methods have locale implementation	debugging through IDE
Security	not existent	not existent	methods can be declared as ”not-offloadable”	not existent

Focusing the aspect of context adaptation, our analysis revealed that only very few MEC applications provide some basic adaptation techniques and would highly benefit from improvement. Regarding the scalability it can be stated that the evaluated solutions perform moderately. In particular, the discovery and integration (S2) could be enhanced to include support for ad-hoc-like opportunistic computing scenarios.

In addition to the mentioned obstacles regarding portability, another important criterion for context-adaptive MEC applications is the availability of the provided services, especially relying on forward error management (A2) and connectivity prediction (A3), which both are non-trivial tasks to carry out and need further research. Ultimately, security is often disregarded in the surveyed solutions, but will probably receive more attention when ready-to-market products evolve from current research efforts.

As already stated in [19], many MEC solutions rely on virtualization approaches. We assume that this is the case, because virtualization is a convenient way to deal with the heterogeneity in MEC scenarios. Solutions to highlight are the *ThinkAir* [9] approach for its good usability achieved by a smart integration into common development tools, only requiring the developer to mark offloadable methods via annotations and allowing a dynamic adaptation to heterogeneous surrogates by making use of virtualization techniques. Compared to the often-quoted *CloneCloud* [8] solution, ThinkAir provides better support for scenarios with a quickly changing context. Another solution to mention is *Cuckoo* [10], relying on the AIDL-interface (*Android Interface Definition Language*) and providing an Eclipse plugin, it appears to be one of the easiest-to-use solutions, performing equally well in terms of adaptation, scalability and portability.

Despite being easy to handle and intuitive, the VM-based solutions all share a common feature: They are not able to fully catch the developers' expertise and have to rely on heuristics to optimize the partitioning and execution strategy which can be a highly complex task when it comes to scenarios with intermittent connectivity where adaptation to future contexts is required. Another drawback is the requirement to synchronize state between the mobile device and the surrogate, which often restricts the offloading to one task (meaning method or thread) at a time in order to maintain consistency and thread safety.

An interesting solution for MEC scenarios that addresses the aforementioned restrictions is a component-based approach presented by Giurgiu et al. [61]. They require the developer to model the application architecture with functional components. Every component of the application has memory consumption, generated input and output traffic, and code size. Using a resource consumption graph, they calculate the optimal distribution of the components. Focusing the important aspect of context adaptation, remarkable solutions to mention are *IC-Cloud* [64] and *Cirrus Cloud* [73] having a strong focus on intermittent connectivity and being one of the few solutions that allow advanced adaptation scenarios by considering both, present and future context.

Summarizing the previous findings we conclude that recent work has concentrated on improving usability, partitioning and scalability issues, but that more complex requirements like parallelization and proper context adaptation still remain open to some extent. Several solutions have been proposed to contribute to the field of

MEC by addressing the presented requirements mentioned in Section 4. However none of the presented solutions are ready-to-use solutions, as they are unable to address all the requirements.

6. CLOUDAWARE

CloudAware differs from previous or similar approaches in the domain of MEC and context-adaptive mobile middleware by its primary design goal to support ad-hoc and short-time interaction with not only centralized resources, but also nearby devices. This idea is extended by the secondary design goal, which is to provide an uninterrupted availability of a mobile application even if no surrogates are available or the connection gets interrupted by using the mobile device as a fallback. It remains the primary instance to hold the mobile applications' relevant state. To achieve this type of spontaneous interaction, classical client-server solutions, service composition, or prominent MCC approaches are often not suitable as they are not able to either cope with the requirements of the ad-hoc interaction or as they are not as lightweight enough to meet the limited resources of a mobile device. How CloudAware faces these restrictions and which general assumptions motivate specific design decisions has been described in previous work [12] and will in the following only be summarized, while the focus of this section is the evaluation of the prototypical CloudAware mobile middleware that is evaluated using real data from the Nokia MDC dataset, introduced afterwards.

6.1. CloudAware Design Goals

To perform offloading, a prominent solution like CloneCloud for example uses a complete image of a mobile device which is running in a virtual machine on a server to execute parts of an application and decides at runtime which threads to offload by using a decision metric based on previous method calls that are analyzed through a profiler regarding their energy usage. The employed heuristics to decide about the offloading further include aspects like the state size, approximate CPU cycles to save, bandwidth, latency, etc.

Along with handling these constraints it often comes to the question how the construed distributed system should behave in the common case of intermittent connectivity, when the connection between the mobile device and the surrogate is interrupted. Following the CAP-theorem it is only possible to achieve two of the three criteria (consistency, availability and partition-tolerance) in a distributed system, but not all three. Hence, in a system with intermittent connectivity one can either go for a shared state between the mobile application and the surrogate, but this requires the mobile application to wait until the connection to the surrogate has been re-established (VM-based solutions, see Section 2.4 and 5.2). Or, to avoid this, it can be more practical to decide which part of the distributed system holds the (primary) state information. To further narrow the selection it is often practical to go for the latter and to only hold the state on the mobile device and just synchronize with surrogates what is necessary at the relevant points in time.

But like CloneCloud, most of the solutions assume a stable connection to the surrogate and do not explicitly handle the effects of link failures. But even state of the art solutions in terms of a fault-tolerant design (e.g., [61]) do not consider the current context in order to adapt migration strategies which are hence often suboptimal and do not allow for dynamic adaptation. As an evolution to the mentioned solutions in the domain of classical MCC other solutions have been proposed that take the effects of intermittent connectivity into consideration, but are often limited to this single context fact.

Instead CloudAware's adaptation architecture has the main goal to be completely generic in terms of the context features that are used for the offloading, so that application developers can concentrate on the business logic and are not required to deal with the effects of the quickly changing context. Nevertheless, they can do so by using annotations or simply weaving the information provided by CloudAware's context monitor directly into their business logic. CloudAware just relies on a common set of context features that are available on almost every mobile device to predict the optimal execution strategy by taking into consideration the success of the execution and the execution metrics (like time, network and energy consumption) to align the execution plan with the global offloading target.

Concluding the previous explanations and resuming the general objectives of MEC applications that have been worked out in Section 2, the specific design goals of CloudAware can be summarized by the following key objectives:

- Speed up computation through parallelization
- Save energy or bandwidth by offloading computations
- Enable offloading for diverse mobility scenarios

This way, we are enabling (future) complex applications on resource-constrained mobile devices by dynamically adapting their execution to changing conditions in their physical and logical environment.

6.2. Runtime Environment

CloudAware uses the principle of "active components" a concept developed along with the *Jadex* [77] middleware. The paradigm of active components unifies component-, agent-, and service-oriented engineering perspectives, to provide a robust and scalable infrastructure in distributed computing environments. Jadex not only ships with several tools to ease the development (like debugging, message inspection, simulation), but also supports - among other features - diverse asynchronous communication styles (remote method calls, transparent service invocations,

DOI: 10.1002/ett
Prepared using ettauth.cls

message passing and interaction protocols) as well as secure messaging, an efficient binary message encoding, the creation of ad-hoc networks, service discovery and NAT traversal over relay servers to bypass firewalls. Jadex has been chosen to serve as a base for CloudAware because it ideally complies with general requirements of mobile cloud environments like distribution, concurrency and failure tolerance mechanisms.

Moreover, the Jadex middleware runs on desktop computers and servers as well as on a broad range of mobile devices, including the Android platform as it requires no modification of the mobile devices' operating system, but just relies on the presence of a Java Virtual Machine that is available out of the box for almost any device.

While carrying out our design decisions we rely on the simple principle to build as much as possible upon widespread commodity software, to leverage the reliability that comes through extensive testing. Hence, we employ standard Java and Android technology, extended by the Jadex runtime environment that we use to implement the functionalities required in CloudAware to perform MEC. For example, to separate front-end user interaction from the offloadable back-end tasks, we make use of Android's service concept by using the *Android Interface Definition Language* (AIDL) specification. This allows developers to easily benefit from the offloading capabilities, while changes to their present applications remain small and are limited to the back-end of the application. Furthermore, this allows the coexistence of regular and cloud-augmented applications as they participate in the standard Android mechanisms like resource scheduling and power management. Hence, no modification of the Android operating system is necessary, enabling the majority of current smartphones to directly benefit from CloudAware's feature set.

Figure 7 depicts a bird's eye view on the resulting execution platform on mobile devices. As already mentioned, Jadex and hence CloudAware support the Android platform. While Jadex runs in an Android process, the CloudAware components are executed in several independent threads. Thereby, we can easily distribute load on multiple processor cores, while Jadex's asynchronous way of execution prevents deadlocks. Finally, MCC applications designed for CloudAware are running encapsulated in specialized CloudAware components, hence developers may just follow an object-oriented approach to implement their applications using the CloudAware API.

Along with this, several further services are required, as mentioned in Section 2.3. These include the Discovery Service, whose task is to integrate the different network interfaces, as well as the Solver and the Coordinator that are in charge of extending the mobile application into the infrastructure. Furthermore, the Context Manager supports the decision of the Coordinator by providing relevant information about the mobile device's context.

6.3. Nokia MDC Dataset

In January 2009, the Nokia Research Center Lausanne (NRC), the Idiap Research Institute, and the EPFL initiated the creation of a large-scale mobile data research. This included the design and implementation of the Lausanne Data Collection Campaign (LDCC), an effort to collect sensor data from smartphones created by almost 200 volunteers in the Lake Geneva region over 18 months [78]. To our knowledge it is the largest dataset that contains information about mobile devices as well as application usage statistics, which is why we chose the Nokia MDC dataset to derive the following information as input to a simulation with the CloudAware mobile middleware:

- GSM/WiFi/Bluetooth state (on/off), discovered MAC addresses and GSM cells, signal strength of WiFi as well as GSM cells, extended with our own measurements to get an assumption about the available bandwidth.
- General information about the mobile device itself: time since the last user interaction, silent mode switch, charging state, battery level, free memory.
- Date, time, location, calendar events, average and predicted remaining duration of stay at the current location.
- Reasoned attributes: remaining duration of stay at the same WiFi AP or GSM cell, user is at home/work, travelling, moving, resting.
- Application usage data: The applications the user interacts with and the specific screens of these applications.

6.4. Generic Context Adaptation Process

One of CloudAware's main features are context adaptation features that have been described in previous work [33] as the *Generic Context Adaptation* (GCA) process and will in the following be briefly summarized. GCA has been designed as a lightweight data mining process that is tailored to forecast arbitrary context attributes out of a provided feature set. More formally, GCA represents a data mining process that provides three types of predictions: A binary classifier (e.g. for predicting the future availability of a WiFi), a multi-class classifier (e.g. to predict a bandwidth range) and a regression mode to forecast real-valued attributes like the execution time of an offloaded task. This way, historical data that has been collected by the mobile device can be used to forecast a future value of a certain context attribute.

To perform a prediction the GCA process needs to be provided with the historical input data in the structure that is used in the Nokia MDC dataset. It is furthermore required to choose the context attribute to be predicted, the context interval (that is required to assign different measurements to a specific time interval) and the forecast horizons (how many time intervals into the future) to initiate the learning phase. The resulting predictive model can then directly be applied to forecast the chosen context

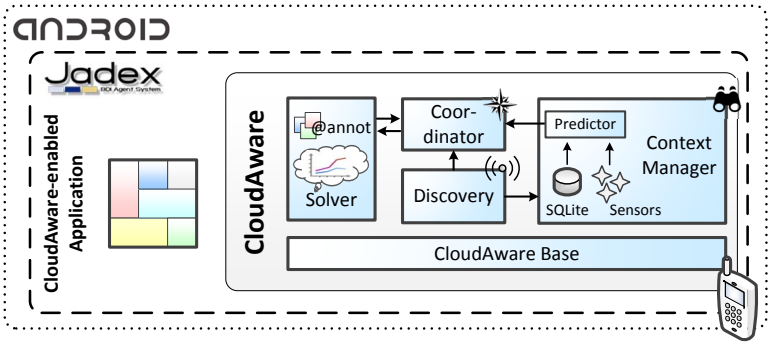

Figure 7. CloudAware: Execution Platform and Architecture [18]

attribute. The data used in the GCA process is mainly the same as in the MDC and is just converted to a representation that is suitable to be presented to machine-learning algorithms. More specifically, GCA runs as a service on top of the Jadex middleware that performs two major tasks: collecting sensor data and providing context for certain context attributes. To perform the first task, the following context attributes are generated, based on the information provided by the MDC dataset:

- The state (on/off) of the different network interfaces (GSM, WiFi, Bluetooth) and the information to which Cell-ID or access point they are connected.
- The current location, mainly in terms of the proximity to certain GSM cells or access points.
- Calendar events and call- and sms-logs.
- The activity of the user (walking/sitting/running).
- State of the mobile device depicted by the time elapsed since the last interaction with the user, the current profile (e.g. silent mode) and the remaining energy.
- Used applications: The information which applications are used and which screens and functions the user accesses.

These features are subsequently used to provide the following forecasts to the CloudAware coordinator that manages the task execution:

- Remaining energy available on the mobile device.
- Probability if a certain task will execute on a specific surrogate.
- Execution time of a certain task on a specific surrogate.
- Bandwidth available to and from a specific surrogate (in a future time interval).
- Future connectivity to a specific surrogate in a future time interval.

To be used together with CloudAware, the GCA process is designed to be executed on a mobile device. As the preprocessing is performed in SQL and the application of the data mining models is performed in Java, the process is able to run on any (mobile) device that provides a Java virtual machine. Nevertheless, the training of the data mining models is computationally intensive and is currently intended to run on more powerful cloud resources, whereupon the trained model is transferred to the mobile device to be used for predictions. We currently use an interval of 5 minutes to collect the required sensor data to not drain the battery too much.

6.5. Evaluation

The evaluation of CloudAware is done qualitatively as well as quantitatively. Within the qualitative evaluation, a descriptive assessment is carried out as to whether the non-functional requirements (cf. Section 4) are fulfilled and whether these could be implemented within the scope of the prototypical implementation. Subsequently, the quantitative evaluation evaluates the nonfunctional requirements by means of collecting certain key indicators in the use of CloudAware. This separation takes place in order to be able to evaluate non-functional criteria such as the performance, which can be determined comparatively simply quantitatively, as well as criteria such as the maintainability of the solution, which are easier to evaluate qualitatively.

6.5.1. Qualitative Evaluation

For the qualitative evaluation of CloudAware, the criteria stemming from the requirements analysis are assessed descriptively and summarized in Table III.

Availability With respect to the error handling CloudAware tries to compensate for failed execution of individual tasks by re-executing them using an adjusted adaptation strategy if execution is regarded promising in the given context (requirements A1 and A2). In order to assess the efficiency of a particular adaptation strategy with regard to the respective adaptation goal and the execution probability of a task, CloudAware uses a series of forecasts, e.g., the future connectivity of a surrogate (requirement A3). For reasons of efficiency and in order to neither overload the mobile device nor the infrastructure, tasks are not re-initiated until their current execution is rated as unsuccessful (requirement A4). However, tasks

Trans. Emerging Tel. Tech. 2017; **00**:1–23 © 2017 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

Table III. Qualitative Evaluation of CloudAware

	A1	A2	A3	A4	P1	P2	P3	P4	P5	P6	S1	S2	S3	S4	U1	U2	U3	U4	M1	M2	M3	SE1	SE2
CloudAware (Concept)	++	++	++	+	+	++	++	+	+	++	+	+	+	+	++	++	++	+	+	++	+	+	-
CloudAware (Implementation)	++	++	++	+	+	++	++	-	+	++	+	+	+	-	++	++	++	+	+	++	+	+	-

that are already running in the infrastructure can only be terminated if there is still a connection to the respective surrogate.

Portability The general ability to integrate the infrastructure into the context-adaptive execution is realized by CloudAware's capability to offload certain tasks (requirement P1). CloudAware thereby ensures that the different tasks are adapted appropriately with regard to the respective adaptation goal in terms of execution time, execution location and implementation (requirement P2 and P3). CloudAware and its system support do not make any exclusive use of the mobile device's resources and integrate as an equivalent operating system process (requirement P4). However, since the system support is not integrated into the operating system, tasks such as the search for available surrogates are only conditionally compatible with the energy-saving mechanisms of the mobile device. Likewise, cross-application coordination of different context-adaptive applications is limited due to the isolation of applications implemented in the Android system platform. This type of deployment, however, corresponds to the usual installation of mobile applications, supports a simple deployment process, and requires only the existence of a Java virtual machine (requirement P5 and P6).

Scalability The overhead, which is essentially created by acquisition and storage of the various context attributes was minimized by restricting the acquisition of context data to certain intervals and determination of available bandwidth to periods in which context-adaptive applications are running. Qualitatively, the overhead can thus be regarded as justifiable (requirement S1). Conceptually, both a proactive and a purely reactive approach to service discovery were developed. However, only a reactive discovery was implemented due to the overhead mentioned above (requirement S2). The concept of task-based adaptation allows to design concurrent tasks, while the correct execution is ensured by using the actor model of the Jadex middleware. Accordingly, without further adjustment of the business logic by the developer, it is possible to perform parallel processing of tasks (requirement S3). However, the possibility of pausing offloaded tasks on surrogates was not implemented (requirement S4) because of the required efficiency and the lower probability of successful task execution.

Usability As demonstrated by the implementation of the example application, the use of CloudAware requires only basic knowledge in component-oriented software development (requirement U1). The specific problems related to mobile cloud computing are largely hidden from the developer (requirement U2). The CloudAware framework integrates into the development process of mobile applications and the corresponding tools (requirement U3). Configuration is done by simply choosing optimization objectives, such as the saving of energy, bandwidth or execution time (requirement U4).

Maintainability Regarding the maintainability it can be stated that the technical control flow of the application is independent of the surrogates used, the execution context or the adaptation strategy and is only delayed within the scope of the adaptation goals - hence it is deterministic (requirement M1). The solution also integrates with the development process and the (debugging) tools for mobile applications (requirement M2). Moreover, only freely available basic technologies are used in order not to depend on a particular service provider (requirement M3).

Security The protection of the confidentiality of information processed by a context-adaptive application can be established by providing special annotations for corresponding components within the source code (requirement SE1). In addition, by using the Jadex middleware, secure communication is ensured within a shared infrastructure and within public networks. Protection mechanisms which are intended to avoid side effects on surrogates by executing non-trustworthy code have not been addressed in the current conception (requirement SE2).

6.5.2. Quantitative Evaluation

The quantitative evaluation is performed by picking 20 users that provided data for at least 18 months from the Nokia MDC dataset and by using the provided context attributes to design an event-based simulation that considers the connectivity (i.e. latency, intermittent connectivity and round trip times) of a mobile device as well as the battery drain, charging state, the limited computational power as well as the energy that is required to compute and to send or receive a specific amount of data via a specific interface (WiFi or GSM). The developed sample application is inspired by the application scenario mentioned in Section 3 and applies different image filters. It produces 20 different types of tasks. For each task the execution time, its variance, as well as input and output sizes (i.e. the amount of data that needs to be transferred if offloaded) have been measured and linked to real application events, that have been recorded in the MDC dataset.

Simulation of the infrastructure The used infrastructure consists of a Samsung Galaxy S5, running the sample application and representing the mobile device. The devices' battery level found in the MDC dataset is considered a baseline. The prototyped sample application is considered to generate an additional battery drain, therefore all tasks could only be successfully executed if the mobile device would be connected to an energy source at all times. As this is not the case, the successful execution of all tasks cannot be achieved.

The cloud is simulated by measurements from and to a virtual server running at a German cloud service provider. To reflect the MEC scenario we furthermore assume cloudlets with the performance of an up-to-date desktop computer to be available at the three most frequently visited locations of each user. Furthermore, due to their limited resources, these cloudlets are assumed to be overloaded at certain times of the day, resulting in longer execution times or even timeouts that lead to unsuccessful offloading.

Scheduling and optimization goal For each of the offloadable tasks, it is decided whether to offload this task to a cloud server or a Cloudlet by predicting its probability to be executed successfully, meaning that the result is returned to the mobile device. All tasks are equally important and therefore they are executed in the order of their invocation time, with the exception that already computed tasks may return to the mobile device with a higher priority. The offloading decision is furthermore influenced by choosing the alternative with the minimum invocation time and if equal the lowest energy consumption.

Description of evaluated scenarios In the following, the different scenarios that are used to evaluate the CloudAware mobile middleware are described. Common to all scenarios is the schedule of application events that is derived from the MDC dataset. What differs for the first three scenarios is the offloading strategy that is mainly influenced by the available infrastructure, derived through the GSM link status and signal strength as well as through the WiFi connection state that is found in the MDC dataset.

Mobile Only: All tasks are being executed on the mobile device. The success rate in this scenario is limited to 86% as the additional energy that is consumed by the sample application produces additional periods of time in which the mobile device has run out of energy.

Cloud Only and Cloudlet Only: All tasks are being executed on the respective surrogate. In both scenarios, the success rate is limited, as sending tasks to and receiving tasks from the surrogate consumes additional energy by using the GSM or WiFi interface of the mobile device.

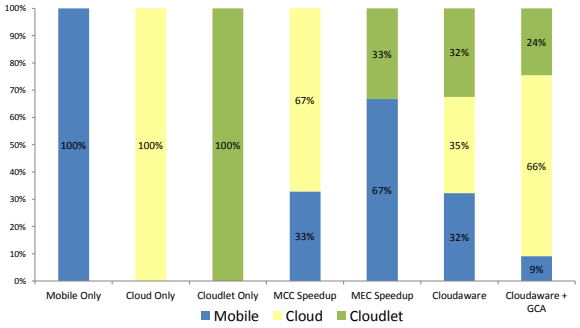

Figure 8. Offloading targets for tasks

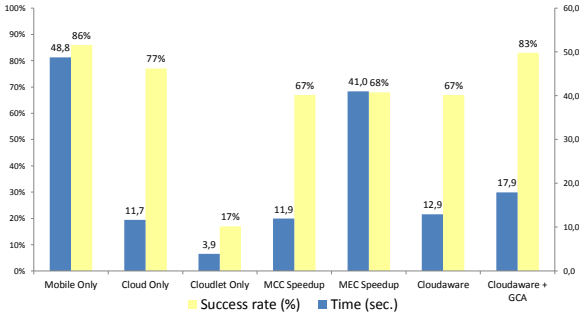

Figure 9. Success rates and execution times

MCC Speedup and MEC Speedup: Tasks are being assigned to the respective surrogate (either a cloudlet or a cloud server) or the mobile device based on the aforementioned scheduling strategy. In this way, tasks are only offloaded if the remote processing of the task on the respective surrogate is expected to consume less time than the local execution.

CloudAware: Tasks are being assigned to the cloud, the cloudlet or the mobile device based on the aforementioned scheduling strategy. Even if this strategy appears equal to the previous one, the higher bandwidth and lower latency to the cloudlets (the cloud servers are connected through a DSL connection in our simulation) can result in higher execution speeds.

CloudAware supported by GCA: Same as CloudAware scenario but the context manager of the CloudAware middleware provides predictions about the probability for a specific task to be successfully executed on a surrogate and furthermore predicts the expected time to execute. This information is used to decide about the offloading target.

Discussion Table IV reflects the results of our simulation, together with Figure 8 and 9 it can be seen that the local execution (Mobile Only), reflecting the baseline,

Trans. Emerging Tel. Tech. 2017; **00**:1–23 © 2017 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

allows an execution of 86% of the simulated tasks while achieving an average execution time of 48.8 seconds. The cloud-based execution results in a success rate of 77% while achieving an average execution time of 11.7 seconds. As expected, the execution time is lower but due to the limited network coverage of the mobile device, the success rate is lower as well. The ”Cloudlet Only” version of this scenario allows even lower execution speeds of 3.9 seconds but the success rate is only 17% due to the limited reliability of the Cloudlet (see simulation setup).

Comparing these baselines to the standard offloading cases in mobile cloud computing (MCC Speedup) and mobile edge computing (MEC Speedup) it can be seen that offloading based on a simple cost function to reduce the execution time is not always able to improve the service quality of mobile users as the success rates are not always higher, which is due to the case that intermittent connectivity sometimes prevents the successful completion of an offloading task.

Compared to this, it can be concluded that already the baseline version of CloudAware provides a speedup of 276% while executing 67% of the tasks. Using the GCA-supported version of CloudAware further reduces the average energy consumption about 15% while it is the only scenario maintaining a similar success rate (83 %) compared to the local execution.

In this way, CloudAware is able to distribute the workload of mobile applications into edge clouds, which allows energy savings as well as significant speedups in application execution for mobile devices. To achieve the savings, CloudAware takes into consideration the context of these mobile devices to allow an efficient and dynamic context adaptation that even anticipates future context scenarios that cannot be foreseen by the developer. Compared to other solutions in this domain, CloudAware does not require the developer to provide any adaptation logic to anticipate this future context but is able to learn the required adaptation logic by using its generic context adaptation features. Moreover, CloudAware is especially construed to deal with the effects of intermittent connectivity and to address the lack of proper context adaptation. In contrast to other related work in this area, the specific focus of this work is on computation offloading techniques that fit into the common development process of mobile applications as well as the developers' mindset.

7. CHALLENGES AND FUTURE WORK

Along with the previously mentioned obstacles, there exist further issues, preventing the extension of current research efforts to market-ready products. Separated into direct practical limitations and open research issues, the main challenges are presented subsequently.

7.1. Current Limitations

Several limitations currently hinder the development and deployment of MEC applications.

Table IV. Simulation results

	Energy (kJ)	Transfer (GB)	Time (s)	Success (%)
Mobile Only	4928	0,0	48,8	86
Cloud Only	7474	17,1	11,7	77
Cloudlet Only	2483	6,0	3,9	17
MCC Speedup	7018	12,8	11,9	67
MEC Speedup	5516	5,7	41,0	68
CloudAware	6300	11,3	12,9	67
CloudAware + GCA	6003	13,0	17,9	83

Limited Network Coverage: The by far biggest limitation is the network coverage. Even if coverage and bandwidth of wireless networks have increased over the years, connectivity metrics are not comparable to those of wired networks. Mainly, the issue of intermittent connectivity brings several restrictions that current MEC applications try to weaken, but which cannot be completely compensated. This often prevents the seamless execution of mobile applications in edge cloud environments [79].

Lack of Centralized Infrastructure: Another aspect is the fact that there is currently no infrastructure of surrogates offered by vendors (except from central cloud providers) that could be used for offloading tasks and no business plan or pricing model exists that could convince providers to establish these. Furthermore, even though highly relevant use cases exist, there still is no high demand for services that benefit from edge cloud augmentation ('killer app') that could justify the investment in such an infrastructure.

Lack of Incentive Models: If users are requested to share their devices with other users, a certain benefit needs to be provided to them to achieve a high acceptance and participation in this kind of user-based collaboration. These benefits can either be monetary or incentive-based (e.g., receiving offloading capacities of others by providing one's own resources). In terms of monetary incentives it is still unclear who provides them and how the business model and its actors might look like. In the latter case, according to [19], it remains uncertain how credit is represented, how the price of resources will be determined and how transactions are processed in a mobile edge cloud environment.

Security Issues: The distributed execution of mobile applications in an untrusted environment is inherently insecure. On a network level distributed services are susceptible to , e.g., denial-of-service and man-in-the-middle attacks, either temporarily disabling the service outright or jeopardizing private data. The MEC infrastructure must also be safeguarded to physical damage resulting in data loss, service manipulation through faulty

VM configurations or privilege escalations resulting in data and privacy leaks [80, 81]. Users need to feel confident about distributing their private data into a shared environment. As found in the performed evaluation in Section 5, security is often mentioned, but seldom considered in the implementation and the prospects for future work. As stated in [23], security-aware offloading can only be achieved in a trusted environment, which can only be realized by implementing a chain of trust on all participating devices of a mobile edge cloud. This not only includes surrogates, but the network infrastructure components as well. Worth mentioning are incentive-based models like proposed in [82] that try to mitigate these restrictions.

7.2. Further Areas of Research

As stated in Section 4 the limitations of mobile devices can be summarized by their computational power, storage and battery life. The approaches taken in the surveyed MEC solutions have already weakened some of these limitations, but the following restrictions are still not solved completely and require further research.

Resource Discovery: The transfer of a mobile application or parts of it requires a fast, dynamic and adaptive resource discovery. As the task of finding and keeping a connection alive can be an expensive process in terms of energy usage, this should be done only when it is required. But when it comes to the point where a surrogate is needed to perform an offloading request, the resource is needed right away and scheduling the discovery process until the offloading request is received already makes some of the offloading tasks redundant. Hence, to align these conflicting targets, an adaptive process is required that dynamically balances the frequency of the surrogate discovery, considering the probability of suddenly receiving the offloading request. Furthermore, as some mobile applications require specific features (e.g., FPGA) to be available on the surrogates, the discovery process should be able to include these non-functional criteria in the discovery process, as discussed in [83]. Ultimately, as mentioned in Section 2 and also found by [35], the quickly changing context of mobile scenarios further complicates this problem as it requires a re-evaluation of the discovery results.

Partitioning and Task Scheduling: The decision for an optimal partitioning- and offloading strategy is still being considered one of the central problem statements in the domain of mobile edge computing [84]. Additionally, the offloading decision also requires to incorporate the mobility pattern as well as the applications' characteristics [85]. The requirement for near real-time data consistency further complicates this problem [5]. Furthermore, the execution of a mobile application should ideally be distributed optimally amongst the available surrogates. To allow this, the application first needs to be partitioned into locally and remotely executable code [35]. This partitioning should first of all happen with respect to the smallest cost, in terms of computation and bandwidth [73, 8]. Second, by maintaining a high level of usability (e.g., remaining responsive), even while the mobile application migrates [54]. Moreover, to achieve an efficient task scheduling, it requires to decide about every potential offloading task (depending on the used granularity, this can be a component, method, thread or even a single line of code) whether to execute it locally or remotely. For remote execution, the decision for a sequential or parallel execution of offloaded tasks (e.g., shared state and task completion dependencies) is another issue to be aware of. While this is a computationally intensive task itself, it needs to be taken care that the expected savings are not overcompensated (compare [7]). With respect to the existing solutions, automatic partitioning and a fine-grained dynamic execution strategy depict an important step towards this issue. Again, the dynamic environment of mobile edge clouds brings further obstacles related to the constantly changing environment, as effects of intermittent connectivity, or more generally a proper context adaptation, need to be taken into account in future work in this domain.

Adaptation: The heterogeneity of mobile edge clouds characterizes an important factor while developing an MEC solution. It needs to be taken into account that their resources, capabilities, and available operating system layers vary strongly. The required adjustments and hence the allocation strategy of the mobile edge cloud solution should take into account these effects and adapt accordingly, but many of the surveyed solutions do not consider the existing heterogeneity.

Lack of Standards: The vast amount of existing solutions might originate from a lack of any standards or a reference architecture, which are both currently missing in the domain of mobile edge computing [86]. The mentioned restrictions prevent companies like network carriers to invest in such an architecture which leads to high boundaries in terms of market entry. But as we could observe with the application stores of Google and Apple, exactly this, providing the right ecosystem (including infrastructure from surrogates to payment), is the key factor to success for MEC. We can conclude that the often cited vision of cyber foraging [16] is still ahead of time, especially due to the mentioned restrictions in the areas of context adaptation, like resource detection, optimal partitioning and proper adaptation strategies - often stated to be the main challenges [35, 83].

DOI: 10.1002/ett
Prepared using ettauth.cls

8. CONCLUSION

As mobile applications demand for more and more resources due to increasing user expectations, new ways to resolve this area of conflict are required. Mobile Cloud Computing as well as Mobile Edge Computing each present a wide range of possible solutions. But despite early standardization efforts [3], there is currently neither an MEC framework nor a cloudlet infrastructure available on the market, which we believe is due to the lack of proper development support and limited context adaptation features that are required to support the broad spectrum of different mobile devices and mobility scenarios.

While much other related work exists ([19], [87], [88]), the purpose of this article was to focus on computation offloading techniques that fit into the common development process of mobile applications, first to support a broad range of mobile services, second to be easily learnable for developers. Consequently, this article first introduced the foundations of MEC together with the typical architecture of an MEC solution. Next, different application scenarios were described to illustrate the practical relevance and exemplify the common characteristics of MEC. Based on these as well as on ISO criteria for software quality an extensive list of requirements was presented. Subsequently, 40 representative MEC approaches from different domains were surveyed and their fulfilment of the respective requirements has been evaluated.

Afterwards, we presented CloudAware, a mobile middleware for MCC as well as MEC. CloudAware faces the challenges by empowering automated context-aware self-adaptation of mobile applications while maintaining features like ease of use, elasticity and scalability. Based on our generic context adaptation process it even allows to anticipate future context scenarios that are unforeseeable by the developer during design time. Besides a qualitative evaluation with respect to the derived requirements we also conducted a quantitative evaluation. In this course, using real usage data of the Nokia Mobile Data Challenge we highlighted the general need for context adaptation in MEC and MCC scenarios and proved our solution to fulfil resource demands of a real-world mobile application in a majority of cases.

Finally, a lineup of open current limitations and further open research challenges has given prospects for future work and research opportunities. Based on these findings our own prospects for future work concentrate on establishing a design guideline and best practices for MCC and MEC solutions as well as on conducting a medium-scale evaluation in the field.

ACKNOWLEDGEMENT

Parts of the research in this paper used the MDC Database made available by Idiap Research Institute.

REFERENCES

1. H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless Comm. and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2011.
2. Cisco Systems Inc, “Fog computing, ecosystem, architecture and applications (RFP-13-078) - research at cisco,” http://research.cisco.com/research#rfp-2013078, 2013, accessed 05.04.2016.
3. ETSI, “Mobile edge computing,” http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing, 2015, accessed 05.04.2016.
4. Cisco Systems Inc., “Cisco pushes iot analytics to the extreme edge with mist computing - rethink,” http://rethinkresearch.biz/articles/cisco-pushes-iot-analytics-extreme-edge-mist-computing-2/, accessed 05.04.2016.
5. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.
6. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
7. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: making smartphones last longer with code offload,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.
8. B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: elastic execution between mobile device and cloud,” in *Proceedings of the 6. European Conference on Computer Systems*, 2011, pp. 301–314.
9. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.
10. R. Kemp, N. Palmer, T. Kielmann, and H. Bal, “Cuckoo: a computation offloading framework for smartphones,” in *Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.
11. G. Orsini, D. Bade, and W. Lamersdorf, “Cloudaware: A context-adaptive middleware for mobile edge and cloud computing applications,” in *2016 IEEE 1st International Workshops on Foundations and Applications of Self-* Systems*. IEEE Explore Washington/DC, USA, 9 2016, p. 6.
12. G. Orsini, D. Bade, and W. Lamerdorf, “Computing at the mobile edge: Designing elastic android applications for computation offloading,” in *8th Joint*

IFIP Wireless and Mobile Networking Conference (WMNC). IEEE Explore Washington/DC, USA, 10 2015, p. 8.
13. A. Hegyi, H. Flinck, I. Ketyko, P. Kuure, C. Nemes, and L. Pinter, "Application orchestration in mobile edge cloud: Placing of iot applications to the edge," in *Foundations and Applications of Self* Systems, IEEE International Workshops on*. IEEE, 2016, pp. 230–235.
14. A. Mehta, W. Tärneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth, "How beneficial are intermediate layer data centers in mobile edge networks?" in *Foundations and Applications of Self* Systems, IEEE International Workshops on*. IEEE, 2016, pp. 222–229.
15. Cisco Systems Inc., "Cisco iox," https://developer.cisco.com/site/iox/, accessed 03.06.2015.
16. M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.
17. D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ser. CHI '99. New York, NY, USA: ACM, 1999, pp. 434–441.
18. G. Orsini, D. Bade, and W. Lamersdorf, "Cloudaware: Towards context-adaptive mobile cloud computing," in *IFIP/IEEE IM 2015: 7th Intern. Workshop on Management of the Future Internet (ManFI)*, 2015.
19. N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.
20. 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, pp. 80–88, 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7060486
21. M. Enzai, N. Idawati, and M. Tang, "A taxonomy of computation offloading in mobile cloud computing," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*. IEEE, 2014, pp. 19–28.
22. F. Berg, F. Dürr, and K. Rothermel, "Optimal predictive code offloading," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2014.
23. S. Goyal and J. Carter, "A lightweight secure cyber foraging infrastructure for resource-constrained devices," in *Sixth IEEE Workshop on Mobile Computing Systems and Applications, 2004. WMCSA 2004*, 2004, pp. 186–195.
24. R. Laddaga and P. Robertson, "Self adaptive software: A position paper," in *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, vol. 31. Citeseer, 2004, p. 19.
25. K. Kakousis, N. Paspallis, and G. A. Papadopoulos, "A survey of software adaptation in mobile and ubiquitous computing," *Enterprise Information Systems*, vol. 4, no. 4, pp. 355–389, 2010.
26. K. Geihs, "Selbst-adaptive software," *Informatik-Spektrum*, vol. 31, no. 2, pp. 133–145, 2008.
27. P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "Composing adaptive software," *Computer*, 2004.
28. M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid, "A comparison of context-oriented programming languages," in *International Workshop on Context-Oriented Programming*. ACM, 2009.
29. T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Workshop Proceedings*, 2004.
30. Inter IKEA Systems B.V., "2014 ikea catalogue comes to life with augmented reality," http://www.ikea.com/ca/en/about_ikea/newsitem/2014catalogue, Aug 2014, accessed 11.04.2016.
31. S. Ladikos, "Real-time multi-view 3d reconstruction for interventional environments," Ph.D. dissertation, Technische Universitt Mnchen, 2010.
32. P. Franco, *Understanding Bitcoin: Cryptography, Engineering and Economics*, ser. The Wiley Finance Series. Wiley, 2014.
33. G. Orsini, D. Bade, and W. Lamersdorf, "Generic context adaptation for mobile cloud computing environments," in *Proceedings of The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016)*, ser. Procedia Computer Science. Elsevier Science, 8 2016.
34. International Organization for Standardization, "ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733, 2011, accessed 05.04.2016.
35. J. Porras, O. Riva, and M. D. Kristensen, "Dynamic resource management and cyber foraging," in *Middleware for Network Eccentric and Mobile Applications*. Springer, 2009, pp. 349–368.
36. G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 5th ed. Boston, USA: Addison-Wesley, 2012.
37. G. Orsini, D. Bade, and W. Lamersdorf, "Context-aware computation offloading for mobile cloud computing: Requirements analysis, survey and design guideline," in *Proc. of The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015)*, 2015.
38. D. C. Doolan, S. Tabirca, and L. T. Yang, "Mmpi a message passing interface for the mobile

DOI: 10.1002/ett
Prepared using ettauth.cls

environment," in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*. ACM, 2008, pp. 317–321.
39. E. Jul, H. Levy, N. Hutchinson, and A. Black, "Fine-grained mobility in the emerald system," *ACM Transactions on Computer Systems*, vol. 6, pp. 109–133, 1988.
40. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2009.
41. T. V. Cutsem, S. Mostinckx, E. G. Boix, J. Dedecker, and W. D. Meuter, "AmbientTalk: Object-oriented event-driven programming in mobile ad hoc networks," in *Proceedings of the XXVI International Conference of the Chilean Society of Computer Science*, ser. SCCC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–12.
42. R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos," in *DCOSS*, ser. Lecture Notes in Computer Science, V. K. Prasanna, S. S. Iyengar, P. G. Spirakis, and M. Welsh, Eds., vol. 3560. Springer, 2005, pp. 126–140.
43. N. Kothari, R. Gummadi, T. D. Millstein, and R. Govindan, "Reliable and efficient programming abstractions for wireless sensor networks." in *PLDI*, J. Ferrante and K. S. McKinley, Eds. ACM, 2007, pp. 200–210.
44. C.-L. Fok, G.-C. Roman, and C. Lu, "Mobile agent middleware for sensor networks: an application case study," in *IPSN*. IEEE, 2005, pp. 382–387.
45. M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *Journal of Supercomputing*, vol. 68, no. 2, pp. 978–995, May 2014.
46. CoDAMoS Project, "CoDAMoS: Context-driven adaptation of mobile services," http://www.cs.kuleuven.be/~distrinet/projects/CoDAMoS/, 2003, accessed 09.07.2014.
47. M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 93–106.
48. W. Zhu, C.-L. Wang, and F. C. M. Lau, "JESSICA2: a distributed java virtual machine with transparent thread migration support," in *IEEE International Conference on Cluster Computing*, 2002, pp. 381–388.
49. R. K. Balan, D. Gergle, M. Satyanarayanan, and J. D. Herbsleb, "Simplifying cyber foraging for mobile devices," in *MobiSys*, E. W. Knightly, G. Borriello, and R. Cceres, Eds. ACM, 2007, pp. 272–285.
50. D. Ayed, C. Taconet, G. Bernard, and Y. Berbers, "CADeComp: Context-aware deployment of component-based applications," *J. Network and Computer Applications*, vol. 31, no. 3, pp. 224–257, 2008.
51. A. Beach, M. Gartrell, R. Han, and S. Mishra, "CAwbWeb: Towards a standardized programming framework to enable a context-aware web," Department of Computer Science, University of Colorado Boulder, Tech. Rep. CU-CS-1063-10, 2010.
52. M. Romn, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: a middleware platform for active spaces," *Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 65–67, 2002.
53. S. Chetan, J. Al-Muhtadi, R. Campbell, and M. D. Mickunas, "Mobile gaia: a middleware for ad-hoc pervasive computing," in *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*. IEEE, 2005, pp. 223–228.
54. G. Ghiani, F. Patern, J. Polet, V. Antila, and J. Mntyjrvi, "A context-dependent environment for web application migration," in *PPD'12: Workshop on infrastructure and design challenges of coupled display visual interfaces*, 2012.
55. P. Yu, J. Cao, W. Wen, and J. Lu, "Mobile agent enabled application mobility for pervasive computing," in *Proceedings of the Third International Conference on Ubiquitous Intelligence and Computing*, ser. UIC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 648–657.
56. K. Lee and I. Shin, "User mobility-aware decision making for mobile computation offloading," in *IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2013.
57. T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt, "AIOLOS: Middleware for improving mobile application performance through cyber foraging," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2629 – 2639, 2012.
58. J. S. Rellermeyer, O. Riva, and G. Alonso, "AlfredO: An architecture for flexible interaction with electronic devices," in *Middleware*, ser. Lecture Notes in Computer Science, V. Issarny and R. E. Schantz, Eds., vol. 5346. Springer, 2008, pp. 22–41.
59. A. Manjunatha, A. Ranabahu, A. P. Sheth, and K. Thirunarayan, "Power of clouds in your pocket: An efficient approach for cloud mobile hybrid application development." in *CloudCom*. IEEE, 2010, pp. 496–503.
60. R. K. K. Ma, K. T. Lam, C.-L. Wang, and C. Zhang, "A stack-on-demand execution model for elastic computing," in *39th International Conference on Parallel Processing (ICPP)*, 2010, pp. 208–217.
61. I. Giurgiu, O. Riva, and G. Alonso, "Dynamic software deployment from clouds to mobile devices,"

in *Middleware*, ser. LNCS, vol. 7662. Springer, 2012, pp. 394–414.
62. X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, ser. PERCOM '03. Washington, DC, USA: IEEE Computer Society, 2003.
63. G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, ser. MCS '10. New York, NY, USA: ACM, 2010.
64. C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik, and E. Zegura, "IC-Cloud: Computation offloading to an intermittently-connected cloud," Georgia Institute of Technology, Tech. Rep. GT-CS-13-01, 2013.
65. T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. B. Heinzelman, "Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *ISCC*. IEEE, 2012, pp. 59–66.
66. A. Grazioli, M. Picone, F. Zanichelli, and M. Amoretti, "Code migration in mobile clouds with the nam4j middleware," in *MDM (2)*. IEEE, 2013, pp. 194–199.
67. M.-R. Ra, A. Sheth, L. B. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *MobiSys*, A. K. Agrawala, M. D. Corner, and D. Wetherall, Eds. ACM, 2011, pp. 43–56.
68. M. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, March 2010, pp. 217–226.
69. X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *MONET*, vol. 16, no. 3, pp. 270–284, 2011.
70. V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg, and B. S. Lee, "μcloud: Towards a new paradigm of rich mobile applications," *Procedia Computer Science*, vol. 5, no. 0, pp. 618 – 624, 2011.
71. H. A. Lagar-Cavilla, N. Tolia, R. Balan, E. de Lara, M. Satyanarayanan, and et al., "Dimorphic computing," Carnegie Mellon University, Tech. Rep., 2006.
72. E. Chen, S. Ogata, and K. Horikawa, "Offloading android applications to the cloud without customizing android," in *PerCom Workshops*. IEEE, 2012, pp. 788–793.
73. C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM Press, 2012.
74. Y. Guo, L. Zhang, J. Kong, J. Sun, T. Feng, and X. Chen, "Jupiter: Transparent augmentation of smartphone capabilities through cloud computing," in *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, ser. MobiHeld '11. New York, NY, USA: ACM, 2011.
75. Y.-Y. Su and J. Flinn, "Slingshot: deploying stateful services in wireless hotspots." in *MobiSys*, K. G. Shin, D. Kotz, and B. D. Noble, Eds. ACM, 2005, pp. 79–92.
76. S.-H. Hung, Y.-W. Chen, and J.-P. Shieh, "Creating pervasive, dynamic, scalable android applications," in *IMIS*, L. Barolli, I. You, F. Xhafa, F.-Y. Leu, and H.-C. Chen, Eds. IEEE, 2013, pp. 43–50.
77. A. Pokahr and L. Braubach, "The active components approach for distributed systems development," *International Journal of Parallel, Emergent and Distributed Systems*, 2013.
78. J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, Do, Trinh Minh Tri, O. Dousse, J. Eberle, and M. Miettinen, "From big smartphone data to worldwide research: The mobile data challenge," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 752–771, 2013.
79. E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 154–172, 2015.
80. R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *arXiv preprint arXiv:1602.00484*, 2016.
81. I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 1–8.
82. S. Abolfazli, Z. Sanaei, A. Gani, and M. Shiraz, "Momcc: Market-oriented architecture for mobile cloud computing based on service oriented architecture," in *Proceedings of Communications in China Workshops (ICCC), 2012 1st IEEE International Conference*, 2012.
83. J. Pauty, D. Preuveneers, P. Rigole, and Y. Berbers, "Research challenges in mobile and context-aware service development," in *Future Research Challenges for Software and Services Conference*, 2006, pp. 141–148.
84. J. Flinn, *Cyber Foraging: Bridging Mobile and Cloud Computing*. Morgan & Claypool, 2012.

DOI: 10.1002/ett
Prepared using ettauth.cls

85. E. Ahmed, A. Gani, M. Sookhak, S. H. Ab Hamid, and F. Xia, "Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 52–68, 2015.
86. Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Communications Surveys & Tutorials*, pp. 1–24, 2013.
87. J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *Journal of Network and Computer Applications*, vol. 48, no. 1, pp. 99–117, 2015.
88. A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 393–413, 2014.