Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 56 (2015) 10 – 17

Procedia Computer Science

The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015)

# Context-Aware Computation Offloading for Mobile Cloud Computing: Requirements Analysis, Survey and Design Guideline

Gabriel Orsini[a,*], Dirk Bade[a], Winfried Lamersdorf[a]

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

**Abstract**

Along with the rise of mobile handheld devices the resource demands of respective applications grow as well. However, mobile devices are still and will always be limited related to performance (e.g., computation, storage and battery life), context adaptation (e.g., intermittent connectivity, scalability and heterogeneity) and security aspects. A prominent solution to overcome these limitations is the so-called computation offloading, which is the focus of mobile cloud computing (MCC). However, current approaches fail to address the complexity that results from quickly and constantly changing context conditions in mobile user scenarios and hence developing effective and efficient MCC applications is still challenging. Therefore, this paper first presents a list of requirements for MCC applications together with a survey and classification of current solutions. Furthermore, it provides a design guideline for the selection of suitable concepts for different classes of common cloud-augmented mobile applications. Finally, it presents open issues that developers and researchers should be aware of when designing their MCC-approach.



Peer-review under responsibility of the Conference Program Chairs

*Keywords:* Mobile Cloud Computing; Computation Offloading; Context Awareness

## 1. Introduction

During the last years, the use of mobile devices has changed dramatically. For instance, todays smartphone users expect to have full access to their own data (irrespective of, e.g., time and space restrictions) and expect to be able to use almost every kind of application on their mobile device. However, despite constant and significant evolution, mobile devices are still and will always be limited in terms of computational power, storage, bandwidth and energy. Furthermore, they will always be less reliably connected and less secure, at least as compared to what we are used to by stationary devices. In order to face these limitations the new generation of mobile applications already relies on, e.g., cloud augmentation as supported by current cellular network standards (3G, LTE) which allows to overcome such intrinsic restrictions of mobile devices. These applications typically comprise computational- and data intensive tasks, like speech- and face recognition, image- and video processing, optical character recognition and augmented reality games, which are nowadays being expected to even run on entry-level mobile devices.

* Corresponding author. Tel.: +49-40-42883-2140 ; fax: +49-40-42883-2328.
*E-mail address:* [orsini,bade,lamersd]@informatik.uni-hamburg.de


Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.07.169

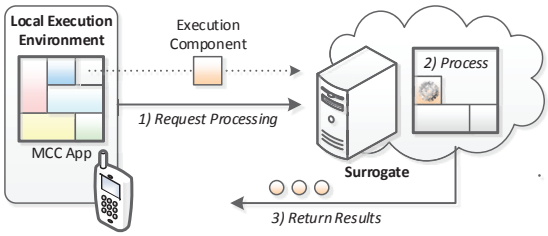


Fig. 1. Offloading Components to Surrogates

In consequence, such types of cloud-augmented mobile applications allow users to perform a variety of tasks and take advantage of the various sensing capabilities of mobile devices as well as to interact with cloud services. Popular examples for that are, e.g., location-based advertising, video-gaming, online-classes and voice recognition assistants. But such solutions are highly domain-specific and the development is challenging and error-prone because of missing tools, guidelines and best practices that ease the process.

Recent work [1,2,3,4] has aimed at tackling the problem of computation offloading and generalizing this approach in order to allow more mobile applications to participate in computation offloading without having developers to deal with the specifics of MCC. However, most of them either lack the support for proper context adaptation or usability. Additionally, there is a multitude of promising approaches that have evolved in the last years, each of them having their specific strengths and weaknesses, what requires a proper guideline to support researches to choose a fitting solution to their problem. This paper summarizes our findings in designing and implementing cloud-augmented mobile applications, and answers the specific research question how to achieve a good software architecture that is suitable for mobile environments, characterized by a quickly changing context. Our contribution in this paper are summarized as follows:

- A structured requirements analysis that serves as a foundation for the evaluation of common MCC-approaches.
- A classification and evaluation of current solutions, highlighting their suitability for specific approaches.
- A design guideline pointing out the relevant criterias to allow researchers to choose appropriate architectures.

The rest of the paper is organized as follows: Section 2 explains the fundamental idea of MCC and characterizes a typical architecture. Section 3 refers to current major problems and develops a corresponding list of requirements for MCC applications, while Section 4 classifies current solutions and compares them with the identified requirements. Section 5 proposes the design guideline and finally, Section 6 draws a conclusion and highlights open issues.

## 2. Background / Typical MCC-Architecture

A mobile application is defined as an application designed to run on a mobile device. But while some mobile applications just rely on the resources and capabilities of the mobile device only, others require the constant connection to distinct cloud resources to provide their full functionality. In between, there is the category of MCC-applications that may be executed entirely on the mobile device, but are able to improve their performance by offloading certain parts onto cloud resources and hence rely on a more dynamic type of coupling.

Figure 1 shows a common offloading task in which parts of a mobile application's code are processed on a so-called *surrogate* and the execution results are returned afterwards. Even if architectural details vary, most of the present MCC-solutions share some common components to enable this type of computation offloading [5]: A *partitioner* that analyzes the application and determines which parts of the code are offloading candidates and a *context monitor* that senses contextual information like available surrogates, battery status and network connectivity. This information is used by a *solver* and the context monitor to decide, whether and on which surrogate to execute the offloading candidates. Finally, a *coordinator* handles additional necessary tasks like discovery, authentication and synchronization.

The offloading decision is considered the main challenge in MCC which involves finding an appropriate granularity for a good partitioning of the application and a context-adaptive deployment strategy to use the right surrogates in a heterogeneous and insecure environment with limited bandwidth and intermittent connectivity. Different paradigms like mobile agents, virtualization and classical client-server architectures are used to support this, while most solutions rely on virtualization[5]. With respect to the general offloading granularity, current solutions can be broadly classified into the following categories: Feature offloading is best reflected by the classical client-server paradigm where a service is receiving a parameterized call and returns the result to the client. Instead, method offloading uses what is commonly known as remote method invocation to perform computation offloading with surrogates. On the one hand, it is beneficial that this paradigm integrates very well with principles of object-oriented design, on the other hand it requires the constant synchronization of shared variables. The remaining categories can be described as virtual-machine (VM) approaches, either employing an application-layer VM (like the *Java Hotspot VM*) or a system-layer VM like *Xen*. The benefit of application-layer VMs is that they are able to abstract from the underlying CPU design (*ARM* is often used for mobile devices, while *x86* is 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).

## 3. Main Requirements

As mentioned in Section 1, the development of MCC-applications performing computation offloading is often complex and requires proper support to ease the development. Models like inter-process communication, remote method invocation or service invocation cannot be employed right away to the domain of mobile computing[6]. This means that not only the problems of classical distributed systems exist, which are the heterogeneity of their components and the requirement of openness, security, scalability, failure handling, concurrency, transparency and quality of service[7], but also the mentioned restrictions of MCC need to be taken into account, being the limited resources, the need of context adaptation in heterogeneous environments and security issues. Additionally, an MCC-solution should comply with general criteria for good software like reliability, usability, efficiency, maintainability and portability as defined in the *ISO/IEC 2501n* criteria for software quality (former ISO 9126)[8]. Hence, we merged the general requirements of these categories with the mentioned issues of MCC as presented in the following.

**Availability:** To perform computation offloading, a reliable network connection is preferable, ideally to all surrogates and with low latency and high bandwidth. Clearly, this is not the case in a real-world scenario where wireless connections typically suffer from intermittent connectivity. Furthermore, simultaneous connections to multiple networks (e.g. Bluetooth, LTE, Wifi) are possible, whereas the complexity of the network infrastructure can no longer be hidden by the Internet protocol (IP)-layer. Providing an overlay network and appropriate metrics to reliably determine measures like bandwidth, latency and the reliability of a specific connection can help to weaken this issue. But even if the effects of intermittent connectivity are identified, errors should not be passed to the business logic, but should be handled by the framework ($A_1$). Ideally, local execution is not the only fallback, but more refined strategies are carried out which employ a rebalancing to other surrogates ($A_2$) or a more sophisticated context adaptation strategy is set in place which includes a prediction of future context scenarios, primarily the network connectivity of the mobile device and the connectivity to current and future surrogates ($A_3$). But ad-hoc scenarios not only incorporate finding new resources, but also require to clean the traces of former use by cancelling offloaded tasks that are not expected to complete in time or that will not be able to return their result to effectively manage the available resources ($A_4$).

**Portability:** Clearly the main and most challenging aspect in the domain of MCC is the offloading decision. To allow opportunistic computing for MCC, a dynamic shifting of computation tasks between mobile devices and surrogates is needed that requires adequate partitioning models that work on different levels of granularity (e.g., methods, classes or components). We consider this the general offloading capability ($P_1$), of which the more fine-grained models are generally expected to better adapt in MCC-scenarios with heterogeneous environments[9]. To allow context adaptation, parameters like bandwidth, energy consumption and the wait time for sending and receiving the code or data to and from the surrogate (among several other criteria) need to be considered in a cost function to decide about whether, where and what to offload. This can make the offloading decision a quite complex and computational intensive task itself, requiring it to rather rely on fast optimization routines or good approximations, to not override the cost savings generated by offloading the task. These tasks should further be carried out automatically and need

to be supported by the application architecture ($P_2$). Some solutions try to move parts of this optimization to the development phase, like in *CloneCloud*[2] where static partitioning is used at compile-time and reassembling of the partitions is carried out at runtime.

Another issue, while distributing computation tasks in a system with intermittent connectivity, is whether and how to handle a global state. While some existing solutions try to always keep a synchronized state for every remote invocation (e.g.[1,10]) others leave it to the developer to synchronize the state between the participating devices[4]. To be able to adapt quickly, while preserving the application state, adequate application architectures and synchronization strategies, supporting the used mobility models, are required ($P_3$). A further aspect for good portability is the coexistence with other conventional and MCC-applications, hence not making exclusive use of resources, but employing a fair scheduling among them ($P_4$). The final aspect is the deployment of the MCC-application. While remote execution techniques like remote method invocation or service calls assume that the code is already present on the remote system, this is not the case in opportunistic computing scenarios. While we assume an appropriate runtime environment to be already present on the surrogate, moving the code of MCC-applications or parts of it to the destination system needs to be carried out directly before starting the MCC-application or even during runtime ($P_5$). Ultimately, the runtime environment should be open to interact with different cloud-service providers and adhere to existing standards to prevent a vendor lock-in ($P_6$).

**Scalability:** Running applications in heterogeneous and changing environments requires dynamic partitioning of applications and remote execution of some parts of it (compare $P_1$-$P_3$). Simply enabling this approach on existing distributed middlewares can bring a considerable overhead in terms of computation and bandwidth usage. Clearly, this is problematic for MCC-scenarios and even worse when no surrogates are available and only local execution of the tasks is performed. Here, the overhead generated by the underlying runtime environment should be minimal ($S_1$). When remote resources are found, they not only need to be provided with the MCC-applications' remote code (see $P_5$), but have to be integrated quickly in terms of rebalancing the workload and eventually providing them with the global execution state. A further task is the integration of particular surrogates providing specialized resources like sensors or FPGAs, that should be supported via a search based on non-functional criteria ($S_2$).

The certainly most important aspect of scalability is the ability to parallelize tasks. Many of the surveyed solutions that attempt an automatic conversion of mobile applications to MCC-applications share the same problem: they lack a support for concurrent execution ($S_3$), as this generally requires the explicit handling of concurrency issues. Another aspect is the efficient management of both, mobile and cloud-sided resources. MCC-solutions generally consider the cloud resources as infinite, which is not the case. Particularly cloudlets have limited resources that need to be taken into account when offloading tasks to them ($S_4$).

**Usability:** Even if some domain-specific approaches already exist (see Section 1) there is neither a common solution nor a cloudlet infrastructure to gather experience on how to design MCC-applications. But as the design and implementation of MCC-applications should be easy and intuitive, domain-specific knowledge should not be necessary and be limited to general aspects of distributed systems ($U_1$). Additionally, an adequate abstraction should hide the implementation details of the MCC-specific routines and provide an appropriate level of distribution transparency ($U_2$). But 'appropriate' does not mean to hide the underlying infrastructure completely as this may prevent exploitation of the full potential of MCC-applications. Therefore, the used solution should integrate with the common development tools, allowing an easy and holistic application of MCC-applications ($U_3$). Finally, the configuration should happen automatically and be based on the developers' domain knowledge (e.g. via annotations), not requiring any configuration on the end-user side, except options to set global targets like: 'save energy' or 'save bandwidth' ($U_4$).

**Maintainability:** An important aspect, very seldom covered in the surveyed solutions is the aspect of maintainability. Software development nowadays is a highly agile and iterative task. To adequately support this task, it is absolutely necessary that the integration of MCC-features does not lead to an increased complexity. This primarily requires the control flow of an MCC-application to be deterministic, no matter which surrogates are used ($M_1$). Offloading a task should always deliver the same results and just different execution times due to the heterogeneity of the available surrogates are acceptable. Another important aspect is the support of common development tools to ease bugfixing ($M_2$). Finally, the architecture should be based on open standards which support the integration of other solutions or platforms and prevent vendor lock-ins ($M_3$).

**Security:** A never ending issue will always be security in cloud computing related to multi-tenancy, concurrency and distribution. Offloading confidential or private information to untrusted surrogates always runs the risk of loosing control over the dissemination of this information. Hence, an easy-to-use instrument is required to distinguish which information may leave the trusted zone (mobile device) and which may not ($SE_1$). Another aspect is the requirement for strict isolation of tasks, that arises from one of the basic principles of cloud computing, the resource sharing. Surrogates need to be able to prevent side-effects of resource sharing and hence need to execute untrusted code in sandboxed environments only ($SE_2$).

Based on the presented criteria the next chapter will deliver a characterization of current solutions, the complete evaluation of over 40 current approaches against the described criteria can be found in our previous publication [11].

## 4. Existing Solutions

The solutions we focus on in our evaluation all share a common criterion, they allow the application developer to stay in "his world", which means keeping the additional effort to learn how to use a solution as small as possible. Because of this, data-oriented programming models are excluded in the following as they do not fit the practices of current mobile application development: *Android* and *iOS* strictly rely on well-known programming paradigms like MVC and object-oriented languages. We further focus on current solutions and mention the older ones just for the sake of completeness. Surveying existing solutions, we classified these according to their nature, in terms of performing the computation offloading. In the following, the six categories will be detailed out:

**Specialized Languages:** There exist quite a lot specialized programming languages for e.g. different domains or specific scenarios, which establish a new or extend an existing paradigm for the sake of more convenient programming. They do so by offering new programming abstractions that ease the handling of certain core aspects of the targeted application environment. Due to these features, application development and in particular the development of context-sensitive applications is eased. But, on the other hand it requires a developer to become acquainted with the new language syntax and programming style and a mobile operating system to support the execution of the language.

**Frameworks and Middlewares:** In contrast to specialized languages, frameworks ease the development by providing a kind of frame in which the developer just has to fill in the application logic. The framework has full control over the execution and calls the application-specific code once appropriate (i.e. inversion of control). Frameworks exist for different programming environments and by using a specific framework, the programmer is bound to that environment, but can use all of the features the framework offers. Middleware platforms (e.g., *Java RMI* [12] also offer programming abstractions for remote execution (e.g., the RMI paradigm) and ease the handling of an application's non-functional requirements. While such frameworks and middlewares are approved ways for realizing distributed applications, it is often difficult to achieve efficent offloading in dynamic environments.

**Distributed VMs:** While the aforementioned solution classes require the partitioning of distributed applications to be done by the developer explicitly, there are also approaches to automate this task. A more or less generic approach are distributed virtual machines (e.g., *exCloud* [10]). These VMs run on several nodes within a network and organize the distribution of application partitions during runtime while still preserving a global application state and handling concurrency and synchronization issues. The high degree of distribution transparency eases the development of applications, but the effectiveness of the approach depends on the choice of adequate heuristics and is restricted to the level of methods or procedures respectively and additionally requires the mobile operating system to support the distribution. When it comes to intermittent connectivity, these solutions are not suitable at all.

**Pervasive & UbiComp Solutions:** There also exist several solutions in the domain of pervasive and ubiquitous computing applications. Approaches like *Vivendi/Chroma* [13] and *Gaia* [14] work on the system level and take care of context data acquisition, resource discovery, data distribution, automatic partitioning of applications and the distributed execution, etc. Using such solutions, distribution aspects of applications can be dealt with already in the design phase and the developer is relieved from lots of lower level implementation issues. But, the effort to become acquainted with such systems is still very high and support is required already at the operating system level. Their main purpose is not the computation offloading, but rather includes moving complete programs from one host to another.

**Native MCC-solutions (non VM-based):** Solutions specifically targeted at MCC can be classified into VM-based and non VM-based. The main idea behind the latter is to simply offload some application tasks onto other devices and hence to unburden the master from e.g. computational intensive tasks. In order to do so, application internals

are analyzed and deployment strategies are created which are applied upon execution of the application (as in e.g. [15]). But the state of the art does not consider the current context in order to create migration strategies, which are hence often suboptimal and do not allow for dynamic adaptation. Moreover, the developer needs to obey certain rules for the strategies to be effective at all.

**Native MCC-solutions (VM-based):** An even higher degree of distribution transparency is achieved by VM-based approaches. *CloneCloud*[2] for example uses a complete image of a mobile device which is running in a VM on a server to execute parts of an application and decides at runtime which threads to offload using a profiler. *MAUI*[1] also profiles a running application to make offloading decisions on the level of methods. The used heuristics include aspects like the state size, approximate CPU cycles to save, bandwidth, latency, etc.

Summarizing the results of the preceding evaluation, it can be stated that usability and maintainability of the surveyed solutions are performing equally well, which can be attributed to the fact that most solutions try to integrate into common development tools and build processes. Nevertheless, the ease of use (U1) could well be improved. Also portability aspects are quite mature. Here, coexistence (P4) and deployment (P5) are receiving top scores, while basic adaptation (P2) and in particular advanced adaptation scenarios (P3) still require further investigation. In terms of scalability the surveyed solutions perform moderately and especially the discovery and integration (S2) would benefit highly from improvements to better support ad-hoc-like opportunistic computing scenarios.

## 5. Design Guideline

Based on the developed requirements and the surveyed solutions we aim to provide a guideline for the developers of MCC-solutions, both frameworks and applications. In our guideline we first describe two common offloading scenarios, centralized and opportunistic, in combination with general design decisions. In the following, we focus on a suitable architecture, partitioning and the offloading itself as well as connectivity issues.

**Centralized Offloading:** The first common offloading scenario, called ”centralized offloading” tries to extend the mobile devices’ capabilities by incorporating distinct cloud resources from dedicated cloud resources (like Amazon Web Services). It is expected that the offloadable code and parts of the relevant execution state are already present on the surrogate. Here, a cpu constraint on the mobile device can easily be solved by simply identifying the critical function and offloading it to a surrogate, similar to a remote procedure call. Most of the surveyed MCC-solutions focus on this scenario.

**Opportunistic Offloading:** The second scenario, called ”opportunistic offloading”, extends the first scenario to the inclusion of all available devices that are ready to act as surrogates. Here, the interaction is way more spontaneous and less long-lasting which makes the offloading decision more complex, as the necessary prerequisites (code- and state-transfer to the surrogate) need to be included in the calculation to decide whether the offloading is beneficial. Furthermore, the remaining connection-time to the specific surrogate might be lower than the duration of the offloading-task itself, likewise making the offloading needless.

**General aspects:** The categories of native MCC- (VM- and non VM-based) solutions depict two good starting points to typical cloud-augmentation tasks as they are designed to address the specific obstacles of this domain. We observed that VM-based solutions perform comparably well in scenarios where little or no context adaptation is required and serve as convincing approaches due to their ease of use by hiding the distribution details from both, the developer and the end-user. However, this high distribution transparency bears several drawbacks in terms of scalability, as only limited multithreading is possible due to comparably high synchronization requirements. Caused by this limitation, VM-based solutions are often restricted to interact with one single surrogate at a time and disallow the interaction between surrogates - we consider them feasible to support the first scenario, the centralized offloading. If the requirements of the mobile MCC-application allow to use a VM-based approach like *ThinkAir*[3], this should always be a first step. If the results are not satisfactory, this is often founded in a high synchronization- and communication-overhead. In this case, a non VM-based approach like in Giurgiu et al.[15] can serve as a base, as it allows to include the developers’ expert knowledge on how the partitioning can optimally support an efficient offloading strategy and to achieve a loose coupling that requires less synchronization efforts to manage the effects of intermittent connectivity.

The latter case is likewise the one to support the second scenario, the opportunistic offloading. Hence the first-order problem in designing MCC-solutions is to decide about the offloading scenario to support, resulting in the decision for an appropriate application architecture. Based on the chosen architecture, a suitable way to partition the workload between the mobile device and the surrogate needs to be established, described in the following:

**Partitioning or the pre-phase to offloading**: Partitioning mainly requires to estimate which parts of the global application state will need to be synchronized during the later offloading, before parts of a mobile service can be executed on a surrogate. Good partitioning reduces the communication overhead between the cloud and the mobile device to a minimum, to make offloading beneficial in contrast to local execution. Again, a component- (non VM-) based architecture often reduces the required amount of computation needed to decide about the relevant partitions as it already represents a certain partitioning. Hence it is often useful to rely on the developers' expert knowledge to decide about a coarse partitioning, while the allocation of these units is handled automatically at runtime[6,16], annotations are a proven practice to capture this information[17]. When designing VM-based approaches, one has to keep in mind that even if a fine-grained and sophisticated distribution like in *ThinkAir*[3] or *CloneCloud*[2] is carried out, the first-order entity of the applications' partitioning still remains an object-oriented approach that is not directly suitable to match the requirements of an optimal distribution. Furthermore, the automated partitioning can easily be overruled by unexperienced developers accomplishing common misconceptions like global variables and other so-called anti-patterns. Hence, it is often beneficial to include the developers' expert knowledge if the results of an automatic partitioning are not satisfactory.

**Offloading:** It is the task of the partitioner to generate bundles of components or other bundlings (depending on the partitioning-level) and to define unmovable subparts (e.g., native methods or device-specific I/O). While the aforementioned tasks generally happen at design time, the subsequent step of the offloading usually happens at runtime. The by far most important step in the offloading process is to efficiently decide which parts to offload. Common ways are either based on solving an optimization problem by performing as in[1] or a graph-based approach as in[15]. The former involves the compute-intensive task of linear programming (LP), which in some cases overcompensates the savings expected by the offloading itself[1]. The latter involves the estimation of a resource consumption graph, where e.g., the vertexes denote the partitions and the edges express the cost of calling them remotely. According to[17] graph-based approaches often perform better, due to their much lower resource consumption compared to LP-approaches. Notable approaches generating graph-based offloading-decisions can be found in[17]. Most notably, Verbelen et al.[9] use an hybrid approach to efficiently achieve an optimal partitioning that can be considered a good starting point for own implementations.

**Intermittent Connectivity and the handling of a global state:** Due to the quickly changing context of mobile environments, static offloading approaches are not able to adapt properly, thus dynamic approaches like in[18] are required. Ideally this involves a context prediction like in *IC-Cloud*[18], especially for long-running tasks. For short-running tasks it is advantageous if the code and data of the function to offload are already available on the surrogate. If this is not the case, offloading is often not useful[16]. Losing the connection to a surrogate poses another big challenge, which is the common problem of leader selection in a distributed system. In a MCC-scenario it is often intended to let the mobile device hold the relevant state information, as implemented in many of the surveyed solutions (compare[2,1]). Further effects of intermittent connectivity like device discovery and link quality estimation can easily be delegated to common infrastructure support solutions for distributed systems. Examples to mention are *Jadex*[19] or *OSGI*, which have successfully been adopted in various MCC-approaches.

To ease the learning processing, we aim to provide some final notes on how to start. An interesting solution that addresses the aforementioned restrictions is a component-based approach presented by Giurgiu et al.[15]. They require the developer to model the application architecture with functional components. 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*[18] and *Cirrus Cloud*[20] 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. Finally, to not start from the scratch, we recommend *Thinkair*[3] or *IC-Cloud*[18] as a good starting point, because an implementation is available open-source and both solutions cover a large share of the requirements mentioned in Section 3.

## 6. Conclusion

Computation offloading scenarios in opportunistic networks have been shown to be a promising approach to overcome the limitations of mobile devices and enhance the user experience. But especially availability, scalability and security in heterogeneous environments are considered highly relevant and complex issues to be targeted, before the vision of unlimited computation power at hands can become reality. Still, there is neither an MCC solution nor a cloudlet infrastructure available so far. It has been argued that major reasons for that are the lack of proper development support and also missing standards to integrate the diverse spectrum of different devices currently present in the domain of MCC. Moreover, the effects of limited bandwidth, intermittent connectivity, and frequent changes of available resources cause further obstacles to the widespread use of MCC so far.

Consequently, this paper first presented an extensive list of requirements, based on ISO criteria for software quality and a set of typical use cases. Further on, several categories of representative MCC-approaches were presented and their respective fulfillment of requirements as identified before was evaluated. Among these, especially the requirement of context adaptation has been highlighted. Finally, to contribute to the development of MCC-applications, this paper highlighted interesting approaches to follow and presented a design guideline for the development of MCC-solutions with a special focus on the main obstacles, namely efficient offloading and proper context adaptation. Yet, there is no ready-to-use solution available in the market, which we believe is due to the lack of proper development support, missing infrastructure and the absence of the ”killer app” relying on cloud augmentation. Furthermore, effective programming abstractions for proper context adaptation are required to unleash the full potential of MCC and will still require further research efforts.

## References


1. Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., et al. MAUI: Making smartphones last longer with code offload. In: *ACM MobiSys 2010*. 2010, .
2. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.. CloneCloud: elastic execution between mobile device and cloud. In: *Proceedings of the 6. European Conference on Computer Systems*. 2011, p. 301–314.
3. Kosta, S., Aucinas, A., Pan, H., Mortier, R., Xinwen, Z.. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: *IEEE Proc. INFOCOM*. ISBN 978-1-4673-0773-4; 2012, .
4. Kemp, R., Palmer, N., Kielmann, T., Bal, H.E.. Cuckoo: A computation offloading framework for smartphones. In: *MobiCASE*. 2010, .
5. Fernando, N., Loke, S.W., Rahayu, W.. Mobile cloud computing: A survey. *Future Generation Computer Systems* 2013;**29**(1):84–106.
6. Porras, J., Riva, O., Kristensen, M.D.. Dynamic resource management and cyber foraging. In: *Middleware for Network Eccentric and Mobile Applications*. Springer; 2009, p. 349–368.
7. Coulouris, G., Dollimore, J., Kindberg, T.. *Distributed Systems: Concepts and Design*. Boston, USA: Addison-Wesley; 5 ed.; 2012.
8. International Organization for Standardization, . ISO/IEC FCD 25000, software engineering software product quality requirements and evaluation - guide to SQuaRE. http://www.iso.org/iso/home/store/catalogue/detail.htm?csnumber=35744; 2014. Accessed 07.07.2014.
9. Verbelen, T., Simoens, P., De Turck, F., Dhoedt, B.. Cloudlets: Bringing the cloud to the mobile user. In: *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*; MCS '12. New York, NY, USA: ACM. ISBN 978-1-4503-1319-3; 2012, p. 29–36.
10. Ma, R.K.K., Lam, K.T., Wang, C.L., Zhang, C.. A stack-on-demand execution model for elastic computing. In: *39th International Conference on Parallel Processing (ICPP)*. 2010, p. 208–217.
11. Orsini, G., Bade, D., Lamersdorf, W.. Survey of 40 current MCC-solutions. http://vsis-www.informatik.uni-hamburg.de/oldServer/teaching/projects/cloudaware/survey.pdf; 2015. Accessed 03.03.2015.
12. Oracle Inc., . Remote method invocation home. http://www.oracle.com/javase/tech/index-jsp-136424.html; 2014. Accessed 09.07.2014.
13. Balan, R.K., Gergle, D., Satyanarayanan, M., Herbsleb, J.D.. Simplifying cyber foraging for mobile devices. In: Knightly, E.W., Borriello, G., Cceres, R., editors. *MobiSys*. ACM; 2007, p. 272–285.
14. Romn, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.. Gaia: a middleware platform for active spaces. *Mobile Computing and Communications Review* 2002;**6**(4):65–67.
15. Giurgiu, I., Riva, O., Alonso, G.. Dynamic software deployment from clouds to mobile devices. In: *Middleware*; vol. 7662 of *LNCS*. Springer; 2012, p. 394–414.
16. Flinn, J.. *Cyber Foraging: Bridging Mobile and Cloud Computing*. Morgan & Claypool; 2012.
17. Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., Qureshi, A.. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications* 2015;**48**(1):99–117.
18. Shi, C., Pandurangan, P., Ni, K., Yang, J., Ammar, M., Naik, M., et al. IC-Cloud: Computation offloading to an intermittently-connected cloud. Tech. Rep. GT-CS-13-01; Georgia Institute of Technology; 2013.
19. Pokahr, A., Braubach, L.. The active components approach for distributed systems development. *International Journal of Parallel, Emergent and Distributed Systems* 2013;.
20. Shi, C., Ammar, M.H., Zegura, E.W., Naik, M.. Computing in Cirrus Clouds: The challenge of intermittent connectivity. In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*; MCC '12. New York, NY, USA: ACM; 2012, p. 23–28.