CloudAware: Towards Context-adaptive Mobile Cloud Computing

Gabriel Orsini, Dirk Bade, Winfried Lamersdorf
Distributed Systems Group
Department of Computer Science
University of Hamburg, Germany
Email: [orsini,bade,lamersd]@informatik.uni-hamburg.de

***Abstract*—The widespread use of mobile devices such as smartphones and tablets is flanked by an ever increasing supply of mobile applications. Along with this trend, expectations and requirements of users rise as well. For example, users do not want to compromise on comfortable daily routines as available on desktop computers. However, an intrinsic characteristic of mobile devices is their limited availability of resources (e.g., CPU, storage, bandwidth, energy) hindering in particular computation-intensive tasks. In this scenario, mobile cloud computing (MCC) promises to overcome these limitations by offering apparently infinite resources in the infrastructure that are transparently accessible also for mobile applications. In order to easily benefit from these offerings, dynamic code offloading has been proposed by several approaches recently. However, such solutions either do not consider the complexity arising from the dynamically changing context in mobile environments adequately or have a steep learning curve inhibiting easy adoption by developers. Therefore, this paper presents a novel approach towards context-adaptive mobile cloud computing. For that, first an extensive requirements analysis was conducted merging ISO standards with users', applications' and developers' needs. Based on this, an evaluation of related MCC-approaches allowed identifying promising concepts as well as current shortcomings. As a result, an MCC-framework, called *CloudAware*, is proposed that eases the development of MCC-applications by offering programming abstractions, multi-level distribution transparency, context adaptation features and is hands-free for end-users.**

I. Introduction

Nowadays, mobile devices such as smartphones or tablets accompany us anytime and anywhere. We get used to not only make phone calls or send messages, but to use increasingly sophisticated applications for a multitude of tasks ranging from intelligent assistants in our daily routine to mobile applications supporting our daily activities. Speech-, face- and object recognition as well as image- and video-processing are among the prominent examples for resource-demanding features contained in such applications [1]. 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. To overcome these obstacles and to allow even more sophisticated applications being used by mobile users, external resources have to be weaved into the local execution of mobile applications [2].

In this way, cloud computing provides, in principle, for nearly infinite resources and makes them available for conventional as well as new (mobile) applications. However, the characteristics of mobile environments require a new approach for dealing with dynamically changing communication qualities, intermittent connections and threats by untrusted counterparts. MCC aims at tackling these challenges, allowing even resource-demanding applications to be used in mobile application scenarios. Here, instead of relying on conventional service calls, MCC-applications mostly adopt the concept of code offloading, by migrating certain parts (code and state) of a mobile application into the infrastructure or onto other nearby devices and collecting the results once appropriate.

Aiming at offering a platform which allows arbitrary applications to make use of generalized mobile cloud computing principles, a multitude of projects arose within recent years. But most of these either have drawbacks in terms of usability or the ability to deal with the ever changing context of a mobile application which is one of the most important challenges to achieve efficiency [2]. Therefore, this paper proposes CloudAware, an MCC-framework that realizes a holistic approach between computation offloading and context adaptation that eases the development of MCC-applications by offering programming abstractions, multi-level distribution transparency and context adaptation features.

The rest of the paper is structured as follows: Section II introduces the fundamental idea of MCC and related paradigms. Afterwards, MCC requirements are subsequently analysed with related concepts of ISO standards and then explained on the background of related work in Section III. Finally, the identified most promising concepts are incorporated in our CloudAware framework which is presented afterwards in Section IV. At the end, we summarize our findings and give prospects for future work in Section V.

II. Background

As the lack of resources of mobile devices is a major problem for many of the aforementioned applications, new engineering paradigms are required. In the following, important views of the problem that received attention in current research efforts along with a typical architecture for computation offloading are presented. Based on this, a common definition of context awareness is given and extended to our view of context adaptation.

A. Mobile Cloud Computing

Mobile cloud computing tries to overcome restrictions of mobile devices and (thus) applications by making centralized

978-3-901882-76-0 @2015 IFIP

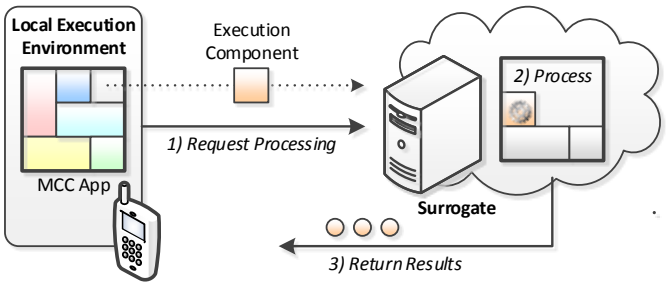

Fig. 1. Offloading of application components to surrogates

resources available to mobile devices. According to [1], MCC is defined as the integration of cloud computing into the mobile environment in order to overcome obstacles related to performance (e.g., battery life, storage and bandwidth), environment (e.g., heterogeneity, scalability and availability), and security (e.g., reliability and privacy). An early definition mentioned in the context of MCC which especially covers the aspect of mobility is the term *cyber foraging* [3]. Introducing the concept of *cloudlets* as an intermediate layer between mobile devices and cloud resources, cyber foraging is expected to further improve latency and execution speed. A similar, but more recent definition is the so-called *fog computing*, defined by [4] as *"[...] a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of network."* The common denominator of the mentioned approaches and research efforts in these domains is offloading computation rather than data. In the following we concentrate on this criterion too, but not solely.

B. Typical MCC-Architecture

Figure 1 depicts a general MCC system perspective: A mobile client running an application may offload certain tasks to be processed on a so-called *surrogate* expecting the execution results to be subsequently returned. Even if architectural details vary, most of the present MCC-solutions share some common components to enable this type of computation offloading [2]: 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.

In a typical MCC architecture the offloading decision is one of the main challenges which involves finding an appropriate granularity for application partitioning 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 [2].

C. Context Adaptation

In addition to the mentioned restrictions of limited resources, the quickly changing environment of a mobile device and hence the integration into a heterogeneous system landscape poses further challenges. Solutions in the field of ubiquitous computing and mobile cloud computing often require an extensive consideration of the users' context in order to quickly adapt to the current situation and use case. Context, in the mentioned sense, is defined by [5] as *"[...] information that is part of an application's operating environment and that can be sensed by the application. This typically includes the location, identity, activity and state of people, groups and objects."* If mobile devices are able to sense and react to their context through physical (e.g. GPS) or virtual (e.g. network connectivity) sensors, they are called context-aware. In order to properly adapt, a more detailed specification of context adaptation introduces further, partially overlapping distinctions [6], namely: parametric adaptation, compositional adaptation and anticipated or non-anticipated adaptation. The last two criteria can be crucial ones for future MCC-scenarios, we therefore consider them main requirements for resource augmentation in environments with intermittent connectivity. Hence, we describe context adaptation for MCC as:

> *The ability of an MCC-application to react to current and future connectivity states and re-evaluate the non-anticipated deployment strategy along with the available resources accordingly by using compositional adaptation.*

Recent MCC-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 the developed approach in Section IV.

III. Challenges and Existing Solutions

In order to develop MCC-applications, various challenges have to be addressed. This section presents an overview[1] of requirements, based on the ISO criteria for software quality and inferred from several use cases.

A. Major Challenges in MCC

As mentioned in Section I, 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 cloud computing [8]. This means that not only the problems of classical distributed systems exist, which include heterogeneity and the requirement of openness, security, scalability, failure handling, concurrency, transparency and quality of service [9], but also the aforementioned restrictions of MCC need to be taken into account, i.e. the limited resources, the need of context adaptation in heterogeneous environments as well as security issues.

Additionally, an MCC-solution should comply with general criteria for good software like reliability, usability, efficiency,

[1]For further details refer to [7]

maintainability and portability as defined in the *ISO/IEC 2501n* criteria for software quality (former ISO 9126) [10]. In an experiment, we considered both the general requirements of these categories as well as the mentioned issues of MCC in order to adapt the ISO requirements to the domain of MCC and then evaluated 40 existing solutions based on the combined requirements catalogue [7] accordingly. In conclusion, the solutions we focus on in our evaluation all share a common criterion: They allow the application developer to stay in his/her world, which means keeping the additional effort to learn how to use a solution as small as possible.

B. Classification of Surveyed Solutions

Surveying already existing solutions, we classified these according to their nature in terms of performing the computation offloading:

Specialized Languages: There exist quite a lot specialized programming languages for different domains or specific scenarios, which establish a new or extend an existing paradigm for the sake of more convenient programming. But, these languages require a developer to become acquainted with the new language syntax and programming style and of a mobile operating system to support the execution of the language. A prominent example is *AmbientTalk* [11].

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. While such frameworks and middlewares are approved ways for realizing distributed applications, it is often difficult to easily transform an object-oriented architecture into a component-based one. Prominent examples are *ASM* [12], *Agilla* [13] and *CoDAMoS* [14].

Distributed VMs: While the aforementioned solutions require the partitioning of distributed applications to be done by the developer explicitly, there are approaches to automate this task. A more or less generic approach are distributed virtual machines (e.g., *Jessica2* [15], *exCloud* [16]). 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. When it comes to intermittent connectivity, these solutions are not suitable.

Pervasive & UbiComp Solutions: There also exist several solutions in the domain of smart home and - more generally - pervasive and ubiquitous computing applications. Approaches like *Vivendi/Chroma* [17] and *Gaia* [18] work on the system level and take care of context data acquisition and resource discovery. 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 idea behind the latter is to simply offload application tasks onto other devices to unburden the master from 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. But the state of the art (e.g., [19]) 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* [20] 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* [21] 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.

C. Results of the Evaluation

Summarizing the previous findings we conclude that recent work has concentrated on improving the usability, partitioning and the 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 MCC by addressing the presented requirements. However, there is no ready-to-use solution, as none of the current solutions is able to address all requirements. As a consequence, we present the architecture of the *CloudAware* framework, an holistic approach to integrate the challenges of both domains: computation offloading and context adaptation.

IV. CloudAware

In the preceding section we presented the discussion of many different solutions employing different levels of abstraction and granularity. We observed that VM-based solutions and distributed VMs 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 multi-threading 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. Even if a fine-grained and sophisticated distribution like in *ThinkAir* [22] or *CloneCloud* [20] 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.

It is also stated by Porras et al. [8] and Flinn [23] that manual partitioning can be more efficient, as application-specific knowledge is included. They argue that the inclusion of distribution details into the original application logic often alters the entire program structure, as it may become more useful to execute an initially linear program as a number of parallel tasks. Furthermore, the automated partitioning can easily be overruled by inexperienced developers accomplishing common misconceptions like global variables and other so-called anti-patterns.

Thus, we decided for a component-based approach to offer both: On the one hand to include the developer's expert knowledge on how the partitioning can optimally support an efficient offloading strategy and on the other hand to achieve a loose coupling that requires less synchronization efforts to manage the effects of intermittent connectivity, which we believe is not possible with totally automatic solutions where handling connectivity issues is the special case, but not the standard case. Using components to serve as a first-order entity furthermore well integrates into the common development process and hence the typical developer's skills and mindset, as they represent a well-known engineering paradigm.

As already shown in the approach by Giurgiu et al. [19] and *μCloud* [24] the concept of self-contained components can be a viable and effective solution to approach the aforementioned obstacles of opportunistic scenarios. However, we differ by the fact that we explicitly try to manage the quickly changing context by considering the mobile devices' context as well as by providing a classification of component- and surrogate types to drastically reduce the complexity of the offloading decision and maintain the same security level like with local execution. Furthermore, we employ the actor model to avoid the aforementioned concurrency issues to further decrease synchronization efforts and allow better scalability.

A. CloudAware Environment

To address the aforementioned issues we introduce *CloudAware*, an MCC-framework based on the "active components" middleware *Jadex* [25]. The active components paradigm, a combination of agent-, service-, and component-oriented engineering perspectives, supports different ways to realize CloudAware's constitutional components (e.g. simple and intentional agents as well as services and workflows). The enabling execution environment Jadex has been chosen because it ideally complies with general requirements like distribution, concurrency and non-functional aspects of CloudAware's targeted application scenarios. Moreover, the Jadex middleware runs on desktop computers and servers as well as on a broad range of mobile devices, 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 all almost any device.

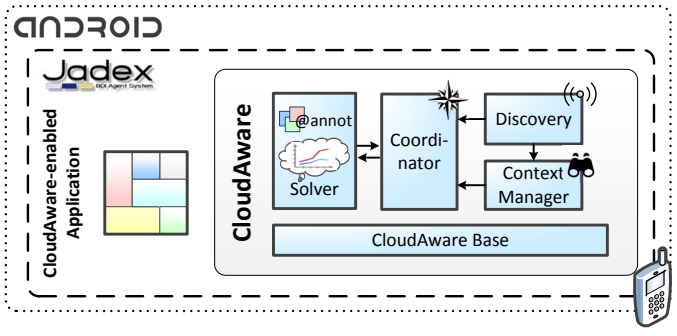

Fig. 2. CloudAware: Execution Platform and Architecture

Figure 2 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. This way, 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 CloudAware components, hence developers may just follow an object-oriented approach to implement their applications using the CloudAware API and optionally may make use of more sophisticated paradigms supported by Jadex, like agents or workflows.

B. CloudAware Components

The CloudAware framework as presented in Figure 2 consists of a runtime environment providing several components to allow optimized offloading decisions. Among these are a *Coordinator* to provide synchronization, a *Discovery Service*, a *Context Manager* providing a (future) connectivity status and a *Solver* that uses component annotations, the Context Manager and further statistics in order to create optimal offloading strategies. In the following, the main responsibilities of the components will be described briefly:

Discovery Service: The current generation of mobile devices provides a considerable amount of communication interfaces (e.g., Bluetooth, WiFi, LTE). Using common mobile device APIs, the CloudAware Discovery Service is able to integrate the different interfaces and provides interoperability over heterogeneous networks by spanning an overlay network that additionally includes MCC-specific cost metrics based on the measurements of the current connection quality and actual surrogate workload.

Solver: Even if the source code is already partitioned into components, a decision on the optimal deployment strategy needs to be carried out, which is the task of the Solver. This includes the basic decision about which components to execute locally and which ones to run on surrogates, as well as a grouping of components with high interaction among each other. Here, the execution time, network latency, bandwidth cost, energy consumption and the amount of input and output data need to be taken into account. But as the context may change quickly, this offloading decision needs to be re-evaluated periodically. It is yet another task of the Coordinator to decide how often to perform this re-evaluation, while not overcompensating offloading savings by the Solver's computations itself. But the Solver's optimization is not limited to a single goal, but on different, even conflicting, goals. While computing an execution plan, the Solver can optimize on energy savings, high performance, bandwidth saving or follow a balanced strategy based on the learned user behavior. Doing so, the optimal execution plan is derived and handed over to the Coordinator. To constantly improve its reliability the Solver periodically performs self-tuning by analyzing past execution plans based on their performance indicators.

Context Manager: The Context Manager provides information about the current and future connectivity statuses to certain surrogates and networks, enabling the Coordinator to schedule long-running tasks while assuring that the results will be delivered. It's function differs from the Discovery Service by the fact that all information provided by the Context Manager is to be considered uncertain due to the mentioned issues of the quickly changing context. To perform computation offloading it is most important to know what latency

and bandwidth to expect to which surrogates. Furthermore it is necessary to know if these surrogates will be reachable and until when or if the connection is expected to be back up later. To provide this forecast on how the connectivity of the mobile device will change over the time, the Solver employs various data mining algorithms from the domain of machine learning in order to perform supervised learning generating prediction models that allow a connectivity forecast. Being computational intensive itself, the model generation is only carried out when the device is connected to a power-source, whereas applying the precomputed models is easily available on mobile devices. In this way, the Context Manager is designed to efficiently deal with the mobile device's limited resources.

Coordinator: It is the task of the Coordinator to merge the information from the Discovery Service and the Context Manager to employ the Solver and to realize the generated execution plans. Furthermore, the Coordinator handles the error management, which involves to recover to a consistent state, even if some surrogates fail or the mobile device suffers from network outages. Currently, the Coordinator handles lost connection states by re-executing tasks locally or on other surrogates. Additionally, the Coordinator takes care of the privacy issues related to computation offloading by enforcing that only components marked as offloading candidates by the developer may be offloaded to potentially untrusted surrogates.

C. CloudAware Component Types

To limit the amount of possible execution plans and to further reduce the complexity of the offloading decision, we introduced categories, both for surrogates and for component types. Having analyzed typical mobile applications that represent candidates for resource augmentation, we found that most features can be broadly classified into two types of operations; idempotent compute jobs and long running non-idempotent operations. For this purpose, we distinguish between three different component types and require the developer to classify the application components accordingly:

Stateless Component: For the class of idempotent compute jobs we envision the *Stateless Component* (SL). The SL is suitable for all tasks that, in case of an error, can recover to the same global state by being recreated by the Coordinator on a different surrogate and by replaying the last service-call addressed to them. As the error handling is performed by the Coordinator the execution of such components always leads to a result if suitable surrogates are available.

Stateful Component: For the more infrequent case of non-idempotent operations we require a different mechanism that is able to properly recover from connectivity failures and re-establish the identical global state. At the same time, this component type, named the *Stateful Component* (SF), qualifies for long-running transactions that are not necessarily expected to complete while connected to the mobile device, as they offer synchronization features to allow disconnected operation. Here, the component that initiates the service-call needs to provide recovery mechanisms on a functional level to compensate the error.

Data Access Object: The third component type is the *Data Access Object* (DAO) that provides an efficient way of virtualized access to the local resources (e.g. sensors or storage) of the mobile device to even allow the offloading of stateless and stateful components that rely on specific dependencies or capabilities of the mobile device.

D. CloudAware Surrogate Types

To further support the Solver in creating the offloading strategy, we also classify the surrogates into three types:

1st Level Surrogates: First-level surrogates are considered to be generally available and well-connected to the mobile device. Examples include the smartphone itself or the user's own stationary computer. Most of the time they are expected to be reliable candidates for the offloading of SL and SF components. Moreover, they are considered a trusted environment.

2nd Level Surrogates: Second-level surrogates differ from the previous type by the fact that their availability and latency is lower, as wireless links (like LTE) are required to reach them. But they depict good candidates for long-running or parallelizable tasks, as their resources can be generally considered high. Hence, they qualify for SL and SF components. Assuming a further service-level agreement with the service provider, even these surrogates can be considered trusted.

3rd Level Surrogates: This type covers all remaining surrogates that cannot be assigned to one of the first two types of surrogates. To support opportunistic scenarios, CloudAware can employ these surrogates for SL components that do not require a trusted environment.

Using the aforementioned classification of components, developers are now required to only annotate their components to indicate which parts of the application's back-end qualify to be offloaded. Tasks like resource discovery, code deployment and the offloading decision are then handled by the framework. Based on the presented taxonomies, a simple use case to demonstrate the different roles is exemplified in the following.

E. CloudAware Use Case

Once a mobile data mining application like face recognition is running for the first time CloudAware tries to offload the stateful (SF) training component onto a trustworthy and powerful surrogate (e.g., a cloud server), where the machine learning takes place and the recognition model is generated. Having synchronized the results back to the mobile device, the succeeding analysis tasks, implemented as stateless components (SL), make use of the previously trained model and can easily be offloaded to all available surrogates that the CloudAware-framework expects to deliver the result in time.

In case of a sudden loss of connection to one of the surrogates, CloudAware re-balances the computation to the mobile device alone and tries do discover new surrogates that qualify for offloading. Based on the forecast of the CloudAware Context Manager and the optimization goal to achieve best application performance, the Coordinator may decide to shift the continuous load to whatever surrogate that is expected to be reachable when the computation is likely to finish.

The depicted architecture easily allows an adaptation to intermittent connectivity and a quickly changing context, while maintaining a seamless application flow. To cover network failures, we assure that there is always a consistent or recoverable

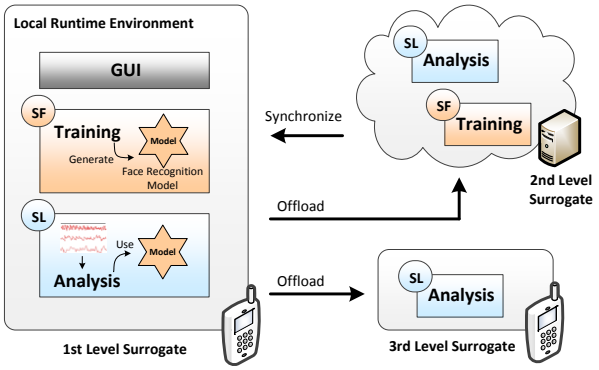

Fig. 3. CloudAware:Face Recognition Use Case

state on the primary (mobile) device. Due to the comparatively low number of components the offloading decision is solved with only a little overhead. Currently, we are implementing selected sample applications like the face recognition app in order to evaluate their performance in real applications, which we plan to present in our future work.

V. 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. Still, there is neither an MCC-solution nor a cloudlet infrastructure available at the market. We argue that this is the case because existing MCC-solutions do not address all relevant challenges. Especially context adaptation is hardly considered, although it can bring a vast asset to future mobile services. Hence, we presented CloudAware as a holistic approach to connect computation offloading and context adaptation. Following a component-based approach, expert knowledge of developers is taken into account to guide offloading decisions. Moreover, CloudAware is especially construed to deal with the effects of intermittent connectivity and to address the lack of proper context adaptation. While many other works exist, the purpose of this paper is to focus on computation offloading techniques that fit into the common development process of mobile applications, primarily to support a broad range of mobile services and, secondly, to be easy to learn for developers. We conclude that to simplify the development, a convenient, but effective programming abstraction is required and that the potentials of MCC can only be unleashed by proper context adaptation, which will be a future source of challenging research problems.

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2011.

[2] 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.

[3] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *In Proceedings of the 1. Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.

[5] 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.

[6] K. Geihs, "Selbst-adaptive software," *Informatik-Spektrum*, vol. 31, no. 2, pp. 133–145, 2008.

[7] G. Orsini, D. Bade, S. Stella, and W. Lamersdorf, "Cloudaware," 2014, submitted to MobiSys 2015.

[8] 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.

[9] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 5th ed. Boston, USA: Addison-Wesley, 2012.

[10] 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_tc /catalogue_detail.htm?csnumber=35744, 2014, accessed 07.07.2014.

[11] 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.

[12] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *J. Supercomput.*, vol. 68, no. 2, pp. 978–995, May 2014.

[13] 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.

[14] CoDAMoS Project, "CoDAMoS: Context-driven adaptation of mobile services," http://www.cs.kuleuven.be/ distrinet/projects/CoDAMoS/, 2003, accessed 09.07.2014.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *ACM MobiSys 2010*, 2010.

[22] S. Kosta, A. Aucinas, H. Pan, R. Mortier, and Z. Xinwen, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE Proc. INFOCOM*, 2012.

[23] J. Flinn, *Cyber Foraging: Bridging Mobile and Cloud Computing*, ser. Synthesis digital library of engineering and computer science. Morgan & Claypool, 2012.

[24] 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.

[25] A. Pokahr and L. Braubach, "The active components approach for distributed systems development," *International Journal of Parallel, Emergent and Distributed Systems*, 2013.