Mobile Computation Offloading: a Context-driven Approach

Matti Kemppainen
Aalto University School of Science
matti.kemppainen@tkk.fi

Abstract

Mobile computation offloading means transfer of execution of mobile software computation outside of the actual device. In this paper, we use a context-driven approach to analyse and design offloading systems. We conduct a literature survey of the types of offloading, supportive architectural models and existing frameworks to propose taxonomies of mobile offloading from viewpoints of an overall computation system and mobile software architecture.

Moreover, we propose offloading into a local network, calling this *local context offloading*. The idea is to minimise network distance to maintain lower latency and higher bitrates, which have an effect to the power consumption in data communications. In addition, our proposal attempts to overcome challenges of scarce Internet connectivity and make good use of collaboration with nearby devices. The main focus remains on the context-awareness. This paper is a step towards a framework that will hide the underlying technology in such dynamic computation environment.

KEYWORDS: computation offloading, software architecture, context-awareness, local infrastructure, battery drain, mobile software performance

1 Introduction

Thanks to open APIs and online application markets, no longer are only device manufacturers and service providers developing and distributing software for the mobile devices. Instead, a myriad of hobbyists and entrepreneurs has welcomed mobile software development. While many mobile devices already have the resources of a late-90s desktop computer, the battery capacities of the devices still remain nearly the same as they were a decade ago. Due to this disparity in technological advancement, only certain types of computation are justifiable in the mobile devices, and they are heavily dependent on the context.

Prior to mainstream mobile data networks, only simple battery drain limitation methods were available, such as optimising the code or hardware and limiting the amount and the speed of computation. In the more recent cases, locally available network connections such as WLANs could also be used, but in the advent of fixed-cost mobile broadband and enhanced connectivity in the most types of networks, moving the computation over the network has become a truly feasible method. Numerous ways of exploiting the well-known client-server model have been available for mobile software developers for over a decade. However, instead of just using the mobile device as a rich renderer for server-generated information, *mobile computation offloading* refers to a more profound change in the system architecture. The key idea is that at least some parts of the mobile software itself are run outside the actual device.

A surrogate is a computing entity that augments the mobile device over a network connection takes over the the computational work of a functionality of a mobile application. In terms of battery usage, offloading proves beneficial if the communication costs for the delegation are sufficiently low and the computation efficiency increase is sufficiently high. Artifical intelligence in a chess game is a classical example of mobile offloading: communicating the situation of the game involves almost negligible cost, and while the painstakingly heavy computation of the optimal move can be done in parallel, efficiency difference to local computing is immense [20]. Therefore computation offloading may provide enormous benefits in certain contexts. What remains is to find a good compromise between the usage of local resources and costs of offloading.

As noted above, energy-efficiency is the keyword in mobile offloading. However, the challenges are not limited to this particular theme. To support offloading of application internals, we need a software architecture fit for the purpose. One example [7] duplicates the state of the mobile device to a virtualised environment in the computation cloud so that there are two (almost) identical environments. However, problems in this offloading model include cost of state synchronization and decision on the appropriate split of computation. We will see that similar challenges are common to many other computation offloading models as well.

Despite the wide variety of viewpoints, we concentrate mainly on architectural aspects in this paper. First, we walk through the problem domain in section two by discussing the taxonomy of mobile offloading. In section three, we identify different architectural models that appear in the existing frameworks and then discuss today's challenges of mainstream computation offloading. Finally, we propose a generic local context offloading scheme and discuss its challenges in sections four and five. While similar idea have been proposed already many times [28, 19, 25], we finally have such technology, which may make it reality.

2 System-Level Taxonomy

As an attempt to give a comprehensive view of the problem domain and its dynamics, we traverse through five high-level viewpoints of mobile computation offloading in this section. Figure 1 summarizes the key factors in each viewpoint. Our

main focus is on describing the potential of mobile offloading. Therefore we intentionally omit deeper analysis of decisionmaking formulae and process-level scheduling.

2.1 Motivation for Offloading

From a selection of recent literature [20, 7, 10, 29], we have identified three general axes of motivation in mobile offloading. *Savings in energy* plays the biggest role of them. The battery capacity has not kept the same pace in development than other hardware components of mobile devices. To prevent excessive battery drain, one approach is to reduce the amount of computation in the mobile device [20]. Whereas energy consumption is a device-centric aspect, *reliability* and *performance* exploit the resources in the cloud.

As a counterweight to the positive factors, several non-technical constraints exist. High performance and robustness involve monetary cost, whereas security concerns require privacy management. Zhang et al. [29] give these two examples as goals in describing the attributes of their cost model. However, rather than bringing any further axes, they act as counterforces to the noted advantages.

Recently, contextual information has been considered only in the role of an environmental factor that may allow or prohibit offloading. In contrast, some earlier visionary papers on pervasive computing consider it as first-class information [25, 28]. While we consider *context* as the fourth axis, it is much more difficult to define than the previous axes. Generally speaking, the more contextual information the mobile device can interpret, the more humane and straight-forward is the user experience [1].

2.2 Nature of Offloadable Functionalities

Many researchers have either noted or implied that certain traits in mobile software embrace towards offloading. *Heaviness of computation* triggers the need for additional resources in order to prevent extensive battery drain [8, 14, 7, 20]. Unlike computational complexity, heaviness may also mean deliberately repeated invocations. *Parallelizability* implies low coupling of the software components and therefore supports computation offloading [29, 9]. *Strength of expression* in terms of length of data reduces the cost of communication [20]. *Time flexibility* makes some context-based optimisations possible [25, 24]. Finally, *state independency* lets us avoid costly synchronization of the internal state [18].

The concepts above contain inherently assumptions from the programming environment. In contrast, Chun and Maniatis [7] give a feature-wise categorization of offloadable functionalities into five classes. In *primary functionality outsourcing*, the offloaded functionality is a computation-intensive key feature of the program, such as speech-to-text processing. This is similar to the client-server applications and must be synchronized (i.e. reactive) from the user's viewpoint. In *background augmentation*, such constraints are less dominating. One of the best use cases in our opinion is photo upload and analysis[1]. In *mainline augmentation*, light-weight computation is offloaded for heavy-weight processing (as described in [18]). *Hardware augmentation* attempts to overcome hardware limitations of the mobile device. Probabilistic analysis and artificial intelligence are use cases for *parallel multiple execution*, which exploits cloud resources to test different scenarios of outcomes that the program may have.

With this categorization, one could think that the data locates in the mobile device. In many cases, however, this is not true. Kumar and Lu [20] bring up the fact with regard of cloud storage services becoming more common. The paper claims that sending the data over the network is not needed that often in future. Instead, just a simple pointer to data in the cloud would suffice. Computation could arguably be done completely in the server-side environment, if the nature of the external data is independent from the mobile software state. However, this is a trivial case from our viewpoint. Therefore we do not focus into this special case. For the rest of this paper, we consider that the offloadable software might need some external data as well, but at least some key elements still locate in the mobile device. A bittorrent client [17] and a database indexer [12] serve as examples that sit shallowly within our definition.

Bridging to decisionmaking, Kumar and Lu [20] describe the classes *never offload*, *depends on bandwidth* and *always offload* to balance the offloading decision with the relationship of computation and communication. Inspired by thoughts in [20, 15, 10, 7], we claim that frequency of computation brings an additional relationship between these two aspects. In a scenario where the same communication feeds multiple occurrences of computation, we can cache and recycle the once transferred data, assuming there are clever ways to perform the needed data transfer. Therefore the earlier classification receives another category called *offload over time*. To recognize its existence, one possible identification method for the existence of such correlation is statistical analysis similar to one presented in [15].

2.3 Decisionmaking

One of the most challenging tasks in mobile offloading is to decide, when it is needed. Generally, offloading is beneficial, whenever the gained efficiency outweighs the costs involved [20]. The difficult part is to define the indicators of efficiency and their related costs, observe their realization and optimise the corner cases where the counterparts compensate each other. One of the key reasons to the difficulty is that mobile environment is much more dynamic than any desktop or server environment. There are generally three different types of decision schemes: static, dynamic and their combination, which we call hybrid. In addition to them, these can be further splitted into automated and human-made decisions.

In terms of energy consumption, static decisions by software engineers have already proven to be very efficient. However, this approach requires careful considerations in the application design. Since static decisions do not take *uneven conditioning* of resources into account, their application is efficient only under certain conditions [25]. Therefore any general categorization cannot provide means to set strict borders. There are certain cases though, where no offloading

[1] Photos would eventually be uploaded from the phone in some way. Therefore it would be convenient to do it in background whenever it is suitable.

motivation	*advantages (axes)*: energy savings, reliability, performance, context *constraints*: monetary cost, security
nature	*embracing traits*: heaviness of computation, parallelizability, strength of expression, time flexibility, state independency *categorization*: software features, general need for offloading
decisionmaking	*inclusion/exclusion*: predefined, dynamic *decisionmaker*: machine, user *timing*: static, dynamic, hybrid *parameters*: static information, hardware resources, network availability, other contextual information
infrastructures	*surrogates*: cloud services, PCs, specialized processors, local environment, other mobile devices *availability*: always available, locally available, context-dependency *selection*: availability, ownership, collaboration, context
networking	*environment*: static, link-level dynamic, upper layers dynamic *mobility*: not handled, external handling, application-specific handling *availability*: always available, known variance, predictable variance, random variance, no connection

Figure 1: Salient points of the discussed system-level views.

should take place. For instance, I/O interactions with the sensors of the device should not normally be offloaded [18]. Also user interace computation is not often offloadable [10], but at least parts of UI can indeed be offloaded [27]. Therefore even these omissions are not categorical.

To tackle the challenge of dynamics, runtime decision schemes profile the code or the environment, or both of them. As an example, Huerta-Canepa and Lee [15] take a statistical approach, in which historical data of the executable's resource usage is combined with the environmental status. In addition to the following general system resources, monitoring includes CPU status, memory consumption, network usage and disk I/O. Dynamic orchestration brings benefits in a just-in-time (or at least timely) fashion. However, profiling takes in turn some resources, and therefore it must be done efficiently. Hence finding an optimum solution is inexact by nature.

Cuervo et al. [10] follow hybrid approach by combining application programmer's vision, static analysis and dynamic environment in the offloading decision. To start with, they collect the methods that the programmer has annotated as remoateble. Calls to these methods may be offloaded, if the system thinks that offloading is beneficial. This way, clear mistakes are avoided and execution is offloaded when needed. On the other hand, CloneCloud project proposes that the system should tolerate mistakes by itself [9].

Selection criteria for the type of decisionmaking is heavily dependent on the nature of the offloadable functionalities. Sometimes offloading may even be the only option, whereas in many cases offloading is not possible at all. Nevertheless, the software developer can easily identify these extremes, thus leaving the real challenges in the middle. The better the context is analysed, the better decisions the user perceives.

2.4 Infrastructures

The words "cloud" and "infrastructure" are used extensively in many recent papers that are related to offloading. The main reason here is to abstract away the physical embodiment of the exploited computation resources. Usually the first notion from such words is a computation service by a third-party provider, for instance Amazon EC2[2] or Windows Azure[3]. In addition to public cloud infrastructures, private cloud deployments (using Eucalyptus[4] or similar software) are becoming more and more common. In enterprise environments, for instance, mobile devices would send their heavier tasks to the surrogates that are in the corporate network, thus complying with the security policies.

Alternatively to the server clusters, the current per capita density of computers allows us to consider offloading to the surrounding devices [7]. Nowadays many specialized hardware computation units are running in the personal computers. In certain types of computation, graphics processing units (GPU) are faster than the main processors (CPU). Despite this, superior computation resources are not always of the utmost importance in offloading. Continuous power supply of a surrogate may as well be the primary target. Therefore also WLAN access points are good for consideration. Moreover, some conventional home routers already resemble minimalistic general-purpose servers[5]. They run Linux, and WRT projects (e.g. OpenWRT[6]) help the tech-oriented owners "optimising" their boxes. Thinking out of the boxes, even cars may support offloading in future, as application programming emerges in their mutimedia systems [4, 21].

Another particular hardware category worth mentioning is simply other mobile devices. Exploiting their resources may feel counterintuitive, because the key driving force for computation offloading is saving their battery life[7]. However, if the devices have a common goal, this setup may save in overall energy consumption. Huerta-Canepa and Lee [16] describe a use case where the subjective exploitation of available resources transforms into collaborative sharing of resources in order to gain utilitaristic benefit. Specifically, a group of colocated mobile phone users form an ad-hoc network, when none of them alone have sufficient resources but

[2] `http://aws.amazon.com/`

[3] `http://www.microsoft.com/windowsazure/`

[4] `http://www.eucalyptus.com/`

[5] For instance, ASUSTeK's high-end router RT-N56U runs a 500MHz CPU and has 128MB RAM. In addition, USB bus enables an easy way to add more hard-disk capacity.

[6] `http://www.openwrt.org/`

[7] We assume here that "two classes of citizens" within the mobile device population is not desirable.

together they would be able to succeed. The benefits of this arrangement are not only technical: in the given use case, the original motivation for collaboration is economical in the form of avoiding data roaming costs. [16]

An obvious problem of the locally available infrastructures is that they are just intermittently (or even just for once) available. On the other hand, remote infrastructures do not have advantage on the issue, because there is no universal guarantee on Internet connectivity either.

2.5 Networking

In order to offload anything, we need a network connection. In an ideal situation, every network node would be globally and unambiguously accessible at all times, with negligible latency and sufficient bandwidth. Especially in mobile networks, this is far from reality. That said, computation offloading papers pay surprisingly little attention to networking. Regardless, Cuervo et al. [10] suggest that both the bandwidth and the *round-trip time* (RTT) between the offloader and the surrogate are a relevant metric in evaluating the cost of network transfers. Chun and Maniatis [8] in turn make a note about the diversity of network environments and effects of mobility.

Besides the challenges of static networks, the need for mobility poses additional routing problems. In a single cellular network, the network architecture masks the mobility on the link layer, allowing the use of any static approach on the logical layer and above. Also link-layer hand-overs behave generally well. However, modern mobile devices support several types of networks; the range can be anything from Bluetooth to Ethernet to WiMAX [7]. According to their availability, the most appropriate interface should be selected for each situation. No network provider can support such switches. [5]

Ra et al. [24] discuss delay tolerance on application level to schedule data transfer in the most efficient way. First introduced in [22], the paper expects the predictability of network availability in order to exploit the context in saving battery life. Sharing the expectations but focusing on the overall throughput, Deshpande et al. [11] carry out a similar analysis using commercially-available networks. Even though the business interest in *mobile data offloading* stems from the congestion of mobile networks, the same solution model, namely using short-range radios such as WLANs, helps both with data surge and battery drain.

3 Mobile Software Architectures

This section discusses some architectural models for the software in a mobile device. As in all architectural problems, the design depends heavily on the way that the designers answer to questions that define the software. Therefore many different approaches to resolve these requirements have been studied. The most important question remains the same: how to support computation offloading?

Many declarative programming languages and their runtime environments have a built-in support for concurrent and distributed computing [13]. However, the covered related work has explored imperative languages (yet with declarative toolsets). Therefore we focus on object-oriented and component-based architectures[8]. The scope of this paper does not allow any further analysis on this aspect.

Level	Offloaded Entity
Feature	sematically coherent parts of the application
Method	method calls (with needed data)
Image	bytecode, program image or volume image
System	low-level code selected by OS scheduler

Figure 2: Architectural approaches.

Listed in Figure 2, our first architectural view, namely *feature offloading*, mandates the mobile software itself to support mobile offloading inhrently. The methods that involve the offloadable computation are themselves responsible of passing the work to the surrogates. On the other hand, *method* and *image offloading* reduce the effects of offloading in application development. Furthermore, feature and method offloading are conceptually different from variations of image offloading and system-level distribution in the sense that the aforementioned focus on the semantical structure of the program, whereas the latter techniques emphasize the supportive role of the computation environment.

In the following, we introduce four frameworks inline with a detailed discussion of the most prominent architectural models. Importantly, instead of falling into only one category, the offloading frameworks proposed so far share qualities of several models. The main goals of the architectures are simplifying or masking the offloading process from the application developer, and providing means to make the offload decisions efficient. We focus on the concepts and therefore–among other things–source building process is mostly out of the scope. While also system-level distribution is definitely interesting, we omit its analysis due to lack of space.

3.1 Feature Offloading

In feature[9] offloading, the mobile software prepares the dataset that is sufficient in solving the computational problem in question. Then the software sends the dataset to the cloud, where the actual computation is done. The mobile software eventually receives the result dataset and acts accordingly. This way, the feature implementation outside the mobile device is rather straightforward, because the interface between the client and the server communicate through an interface that abstracts away the computation environment.

With this definition, applications making use of feature offloading range from World Wide Web to graphical desktop sharing (VNC). This scheme is particularly useful, when both the offloader and the surrogate share the same long-

[8] Paradigmatic purity is not of key importance here; we do not attempt to differentiate the paradigms more than appropriate.

[9] In this context, *feature* means a logical composition of software execution that the user senses as a distinct and atomic behavioural trait.

term storage for data[10]. Since client-server model is the most prevalent communication method in today's Internet and RPC, REST, SOAP and similar access protocols are well-known, wide deployments of the applications exploiting this model are easy.

As noted in 2.3, efficiency is potentially very high for the statically offloaded features (e.g. a Google search). As a counterweight, by our definition, the application must prepare the transferrable dataset by itself. In order to maintain efficiency, the more complex the offloaded feature is, the more carefully the software must be designed. Therefore decisionmaking is almost inherently manual by the programmer in the architectures relying on this model. However, the application developer has consequentially the control over offloading, and therefore deployment of this offloading type is rather safe. In addition, this type of offloading does not have any (direct) effect to those funtionalities that are not crafted for offloading.

Despite its manual nature, feature offloading does not necessitate static timing decisions. *Cuckoo* framework requires a developer to write an interface for the offloadable feature [18], and thus has qualities of feature offloading. *Elastic Application Model* introduces a similar approach by requiring use of so-called *weblet containers*, which encapsulate an independently[11] runnable part of the software ("a weblet") [29]. Both of these frameworks are designed for dynamic timing of offloading.

3.2 Method Offloading

Method offloading stands for transfer of execution of subroutines. It is conceptually somewhat similar to feature offloading. The difference is that the computation split of the code happens on a semantic rather than logical level. Therefore offloading decisions may also be based on purely computational statistics. Also some distributed object frameworks fall into this category.

In order to make method offloading attractive from the developers' point of view, programming environment (compile-time or run-time) should be somehow aware of its own structure and be able to use this information in offloading decisions. In practice, *introspection* and *method wrapping* are required features from the programming languages and their runtime environments. Cuervo et al. [10] rise to the challenge with a method-level offloading framework called *MAUI*. With help of .NET Reflection [12], which is one way to do code introspection in CLR[13], MAUI wraps during the compilation the method calls that the programmer has tagged as candidates for offloading.

Even bigger question in this model is what information surrogates initally need. One alternative would be to offload initially nothing, insisting that accessing objects requires communication over network [14]. MAUI's approach represents the other end: the visible state of the program is offloaded in its entirety. However, the developers of MAUI acknowledge this as a point of naïveté in their system and are working on a more fine-grained solution [10]. Finally, Cuckoo has taken a stateless approach and effectively leaves the synchronization of bigger datasets to the application programmer [18].

Method offloading does not conceptually require any similarity between the offloader and the surrogate, albeit some existing frameworks (e.g. CloneCloud [9]) are designed to work on the identical sandboxes. Since method calls are usually references instead of code transfer, they may be bound in another environment with a different implementation (e.g. Cuckoo [18]). Whereas identical environments open up an easy way to type safety, robust *object broker* architectures such as CORBA may compensate the downsides. The general downside of method offloading is that the programmer is restricted into certain programming languages or frameworks that support identification of methods and their wrapping.

3.3 Image Offloading

Image offloading is an intuitive yet difficult-sounding approach towards offloading computation. Simply put, the image of the program code is offloaded to a virtualized environment and the state of the machine is maintained to correspond to the one of mobile device's process.

There are many levels of image offloading. System virtual machines (VM) allow many instances of operating systems to share the resources on one physical machine. The processor runs natively the machine instructions of a hardware-assisted or *native* VM. However, this poses a problem in our context, because offloading the operating system goes down to machine instructions. Most of the mobile devices have an ARM processor, while servers and desktop are usually based on x86 instruction set architecture (ISA) [10]. Software-driven or *hosted* VM may emulate any ISA, but the efficiency is inferior to the native execution.

As another aspect, application-layer or process VM refer to a virtual environment similar to hosted VM with the exception that it abstracts away the underlying operating system and hardware for one process at a time. One well-known example of such platform is Java VM. As an input, the VM takes an application in a special *bytecode format*, which is an optimized binary presentation of the source code. This arrangement provides a certain level of platform-independency to overcome the challenges of differing ISAs.

CloneCloud project has customized Android's application-layer VM (Dalvik) to support computation offloading. It also attempts to prevent unnecessary suspension of the process by working at thread granularity. [7, 9] In contrast, the reference implementation of Elastic Application Model runs only the offloaded weblet under the management of a cloud-side controller.

3.4 Challenges in Today's Frameworks

Considering mainstream deployment as the goal, the biggest problem with MAUI, CloneCloud, Cuckoo and Elastic Application Model is that they are simply not available. No

[10] Described in section 2.2

[11] It is worth noting that these weblets may run other weblets, effectively producing a tree hierarchy.

[12] `http://msdn.microsoft.com/en-us/library/f7ykdhsy(v=vs.71).aspx`

[13] Common Language Runtime, Microsoft's runtime environment for its Common Intermediate Language (CIL).

software developer can start adapting to them, nor the mainstream mobile device users may feel their benefit. Even if the frameworks were available, the suggested frameworks would require awareness from the developers. Albeit the presented solutions already lean towards easiness in application integration, they propose certain approaches towards software development, to which the developers would need to adapt in order to gain the benefits. Therefore the existing applications might need an extension or a partial rewrite to introduce compatibility with the frameworks. At least the source code must be rebuilt for static code analysis.

Applicability of the frameworks remains questionable as well. Whether they worked in anything generic is not at least mentioned yet. The examples given in the experiment reports mainly cover some specifically-crafted use cases for mobile offloading. No analysis has been done with such software that is commonly available. We suggest that this is partially because of the recognised need for adaptation. In addition, this leaves the current guidelines for the software developers quite general, since the corner-cases are not discussed.

Diversity of the approaches is also problematic. Every framework has its own environmental needs and uses their own application-layer networking protocol. This trend is also likely to continue. Because offloading is application-specific, a mobile device owner would probably use a number of offloading frameworks due to multiplicity of applications. From the viewpoint of cloud service models, this would be a strength. Application publishers would provide a surrogate, or separately available offloading services might emerge. In either case, the user would be tied to the available servers and service providers, unless installation and resource consumption of the surrogate software is carefully examined prior to deploying a framework.

As a final note, exploiting only one aspect of offloading would easily make the system naïve. Today's proposals scatter to many viewpoints in order to gain more benefit from the context. However, excluding Elastic Application Model, they omit user interaction as input. As trust issues are yet to be solved, this is probably a point of future work. Even so, we argue that the need of user interaction exists. The extent of it depends on the shape of the cloud, which partially defines the user's vision of trust (to service providers) and personal needs (in terms of monetary cost) [29].

4 Offloading in Local Context

In this section, we discuss the architectural concerns regarding computation offloading to locally available surrogates. Our definition of local offloading thus defines the network distance. In addition to the mobile devices, local-area networks have many other citizens. As described in section 2.4, they range from ordinary consumer-level technology (such as WLAN access points) to sensors and low-end servers. Moreover, the number and diversity of devices are continuously increasing. While music player integration in the car audio systems has been around for a while [2], car-specific applications using built-in touchscreen consoles is on the edge [4, 21].

To many of these surrounding devices it applies that their energy reserves are virtually infinite, as they are connected to a power grid or an aggregate. Presuming their capability to general-purpose computing, their resources may therefore be harnessed to offering a relief mechanism for the surrounding mobile devices[14]. Motivation to this lies in the basic principles of telecommunication: the longer the distance, the higher the latency. Earlier research has shown that offloading over high-RTT connections is expensive to the battery [10]. The following collection of use cases illustrates the potential.

> *Let our imaginary person Oliver have a bunch of different networking-enabled devices that are fit for generic computation. He wants to use these devices to help his mobile phone save energy. When Oliver is at home, the computation is offloaded to his WLAN access point, multimedia system or even a futuristic coffee machine. During a car journey, the phone communicates mainly with the car, using minimal computation resources of its own. When at work place, a local augmentation solution helps Oliver's phone to survive the day's tasks. On the weekend trip to the countryside, the train offers an Internet uplink and a local computation infrastructure. Since Oliver is keen to following the news, he launches an instance of news subscriber application on the train's surrogate. Other travellers may join to follow the same news, thereby reducing the use of uplink capacity. Finally, Oliver wants to work on the photographs with his tablet after returning from the trip. As his home multimedia system processes the photos, the tablet is just a conductor of the infrastructural orchestra and an interface to the user.*

In terms of software functionalities, this example revealed numerous possibilities for offloading to locally available *hotspots*. In the car, the owner of the mobile device might be driving, and thus only background augmentation would be realistic. In many such situations (e.g. virus scan) a significant amount of data is located in the device itself. Therefore use of a nearby device would be beneficial. On the other hand, while collaborative operation in the train would be in the interests of the train company, it would also save the battery lives of the mobile devices by providing better RTT and transfer speed than accessing the news service directly. This in turn would improve user experience. Lastly, physical essence would not limit the feature set of a compact device. In addition to saving uplink capacity here as well, offloading to a nearby device would put otherwise wasted resources into good use.

As the future prospects given above sound promising, the development of mobile offloading has yet to overcome many technical challenges. In the following, we discuss a few of them. Due to the reasons already mentioned in our discussion about the diversity of the offloading frameworks (section 3.4), the most important view in answering to these problems is to thrive towards common standards. Finally, we peek quickly into the business domain.

4.1 Service Discovery

Local surrogates need to be *published* and *discovered*, before their use is possible. After discovery follows *resolution* of

[14] The surrogates would maintain their original tasks.

the available services. *Universal Plug and Play* (UPnP) is a protocol suite for these routines in a local IP-based networking environment such as WLAN. UPnP over Bluetooth is an alternative to defeat the limitation of native Bluetooth device discovery, which ranges only to the Bluetooth piconet. UPnP would enable use of Bluetooth as a gateway in extended service discoveries [3]. However, among other benefits over UPnP, Apple's *Bonjour* has superior API support and extensive documentation. It also supports a form of infrastructureless domain name system (DNS) called *mDNS*, which uses IP multicast. [6]

Kemp et al. [18] complete the service discovery with QR codes[15], which would contain the address of the server. However, this solution requires a unified naming scheme. URLs work only on a logical networking layer and imply existence of a working network connection. This would lead to a problem in an environment, where the surrogate is only available through a certain link-layer connection. Nonetheless, occurrence of such situation is unlikely due to high level of IP penetration in the devices capable to generic computing.

Due to these facts, we suggest that IP together with mDNS is sufficient in terms of naming scheme requirements. As Bonjour is already a widely-deployed industry standard, it is a natural choice for zero-configuration use of local context offloading services. For longer-distance surrogates that in reality would be used every now and then, the mobile device might just maintain a simple list of addresses to the user's subscriptions.

4.2 Code Transfer and Execution

Remote method invoking technologies typically assume that the remote implementation is already in place. In the dynamic environment described above, this is not the case. Therefore we need a mechanism to transfer the program code to the surrogate. There are generally two alternatives to tackle the issue. Elastic Application Model (section 3.1) takes a *bundling* approach [29]. In other words, it includes the needed program code to the information package that is sent to the surrogate. Another way would be to *reference* the needed code, when the surrogate would have to retrieve it.

Both means have good and bad sides. First, the referenced code must be available, whereas bundling approach works even without Internet connectivity. An easy way to circumvent the problem is to keep one copy of the offload code in the mobile device as the backup source. However, this would in turn prevent potential environment-specific optimisations available for downloading, since the mobile device cannot be expected to carry all of them. Second, surrogate may cache the static parts of popular weblets. Thus transfer is not always needed. In any case, code transfer should be handled as soon as the need is evident, in order to prevent delays or other artifacts to the user.

After the surrogate has received the program code, there must be a way to execute it. Today's mobile devices are diverse in terms of programming languages and runtime environments. Therefore supporting all of them is a real challenge. This time we face three alternatives. The first one is to support such a selection of platforms that is satisfactory to the target group of the surrogate. This would allow us to run the same code in the surrogate as in the mobile device. Another way around is to have another implementation specially crafted for the surrogate's software stack. Needless to say, this would cause overhead for the application developer. Furthermore, software dependencies are bound to produce problems in differing computation environments [23], and therefore a strict independency clause is likely to be concentrated on the offloaded code. Third, the abstraction level of offloaded procedures might be maintained so high that the computation is possible in every environment.

Finally, the user is not alone with trust issues. In our concept, surrogate's environment is about to run third party code in an automated fashion, for which we need security management. The following two places are probable for offloading purposes. First one is infrastructure services such as public clouds or local data processing units. In them, computation is already sandboxed with virtualization techniques or restricted execution environments.

In this light, the most likely transfer model for local context offloading is a combination of bundling and referencing so that the offloader is able to provide a baseline implementation for the surrogate, if no optimised version is available. Whereas making assumptions of the virtualisation or sandboxing level requires more empirical data, a good starting point would be to consider one offloaded feature at a time, focusing on the modularity of the system. Therefore we suggest Elastic Application Model as the basis for future considerations. In further analysis, other offloading frameworks may be worth a revisit.

4.3 Cloud Transfer

In a mobile environment, local context changes frequently. Therefore available computation services change as well, and computation must occasionally be moved from a surrogate to another. There are two possible routes in the transfer: directly between the surrogates or using the mobile phone as an haulier. Nature of offloaded functionalities plays a key role in this selection. Some background tasks could continue in the cloud and then be moved nearby the mobile device, when there are sufficient resources or a specific reason for it. In terms of energy consumption, even better solution would be a proxy model, where the computation remains in the public cloud [29, execution pattern no. 3].

Connectivity may not always be instant, and therefore temporary data storages are needed. In order to make use of them, delay tolerance must be known or estimated. In addition, reliability of local surrogates is dependent on the service provider, and it is difficult to imagine any service-level agreement in such computation environment. This creates an inherent need for checkpoints to resolve any data losses.

4.4 Business Opportunities

All the examples in the previous sections assume some kind of access to surrogates. Here we substantialize our views of getting such access to publically available resources. To start with, privilege to use hotspots would be sold as a com-

[15] `http://www.qrcode.com/index-e.html`

plementary service, e.g. together with a train ticket. Second, hotspots might run some special services that are configured to the needs of the local environment. Examples of such locality-awareness are virtual guide in a museum or flight tracker at an airport. The needed client-side software would be easily available from the surrogate. Next, if several users want to run the same program on a hotspot and the data is shareable (e.g. subscribing a public RSS feed), the effort would be combined into one. Such services would also enable advertising with sponsored offloading of applications, keeping them free of charge to the users.

Equally important is to embrace the businesses and even individuals to providing computation resources. For instance, a homeowner association would like to offer local augmentation and Internet connectivity to the public in a residential area. To overcome its maintenance fees while maintaining openness, the service would not be free of charge. If the payments are charged in virtual currency, only the banking service provider would need the actual credit card information. Since such virtual banks set their own rules to the businesses, it might be easier to get started with hosting a computation service than with help of traditional banks whose operations are strictly regulated by laws.

Also collaborative wireless networks such as Fon[16] or Wippies[17] are already daily business, computation hotspots have also potential to similar deployment scenarios. If this kind of offloading scheme can provably reduce overall energy consumption from the system, even public institutions such as municipalities[18] could be interested in setting up hotspots.

5 Future Work

This paper only scratched the surface of local context offloading. At least the following challenges are yet to be surveyed in this delicate domain.

Technical Aspects In addition to future challenges in section 4.2, we propose further research on the following topics. Starting with *Unified configurations of surrogates*, the feature set of the surrogates should be universally comparable, so that profiling the surrogates would be possible by the manifest of it alone. This leads to *quality of service guarantees*, which are important in determining the benefit/cost balance of a single offload decision and adapting the other parts of the software to the usable resources.

Moving to advanced application areas, *Cloud of hotspots* refers to a grid-alike arrangement of the nearly positioned hotspots and enables sliding the offloaded computation in the grid as the mobile device is moving. Furthermore, *process transfer* allows an instance of multi-platform application to move from a terminal device to another, as demonstrated in Aura project [26].

Information aggregation frameworks (such as already existing mobile location APIs) make use of contextual information easy by reducing the raw data into an informational and purpose-driven format. For instance, a certain level of location-awareness may be achieved this way. However, it is questionable whether a more precise context may be defined by applying the data to create a more general contextual model.

Social Aspects A good number of mobile offloading papers considers trust issues as a theme for future work, and maintaining confidentiality in cloud envinroments is under active research. The dynamic nature of local context offloading even increases the level of burden. In addition, the collaborative application concepts may require certain anonymity of the handled information and its users.

Business Models As started in 4.4, local context offloading may have potential to change our IT ecosystem. One of the biggest questions from the developer's point of view is the openness of the environments. In order to gain a critical mass, a business-backed offloading model is probably a necessity. On the other hand, if the development work is entirely enterprise-driven, we may end up to *a walled garden* or some other form of vendor lock. In our opinion, realising local context offloading to its full potential requires open and innovative business environment.

6 Conclusion

In this paper we formed a taxonomy of mobile compution offloading by integrating existing information from related work and proposing our own supplements. This was done from five aspects in the system-level concepts, whereas four levels of offloading-enabled mobile software architectures were identified. We also discussed a small number of recent efforts in compution offloading framework and analysed their current potential in mainstream usage. Lastly, our vision of local context offloading showed a variety of ways to improve the user experience and energy efficiency in the mobile device. In order to make the vision reality, we selected an existing basis and proposed some directions for future research. Throughout the paper, we maintained that contextual applicability brings challenges, but good answers may bring unforeseen possibilities to mobile computing.

References

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, 1999. Springer-Verlag. ISBN 3-540-66550-1.

[2] Apple Inc. iPod your BMW: Apple & BMW unveil the first seamless integration of iPod and car audio system. June 2004. URL `http://www.apple.com/pr/library/2004/jun/21bmw.html`. Press release. Retrieved April 5, 2011.

[16] `http://www.fon.com/`

[17] `http://www.wippies.com/`

[18] Wireless networks run by municipalities are already common in Finland and many other countries.

[3] A. Ayyagari. *Bluetooth ESDP for UPnP*, 1 2001. URL http://www.comms.engg.sussex.ac.uk/fft/bluetooth/ESDP_UPnP_0_95a.pdf. Technical Draft. Retrieved April 28, 2011.

[4] M. C. Baca. Tesla CEO: Model S will support third-party apps. March 2011. URL http://venturebeat.com/2011/03/16/tesla-app/. News report. Retrieved March 24, 2011.

[5] F. Bonomi. The future mobile infrastructure: challenges and opportunities. *Wireless Commun.*, 17:4–5, October 2010. ISSN 1536-1284. URL http://portal.acm.org/citation.cfm?id=1921927.1921929.

[6] S. Chesire. *How does Zeroconf compare with Viiv/DLNA/DHWG/UPnP?* URL http://www.zeroconf.org/ZeroconfAndUPnP.html. Technical report. Retrieved April 28, 2011.

[7] B.-G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th conference on Hot topics in operating systems*, HotOS'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[8] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 7:1–7:5, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0155-8.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. EuroSys 2011, Accepted Papers, 2011. The conference takes place April 10-13, 2011.

[10] 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*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5.

[11] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3g and metro-scale wifi for vehicular network access. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 301–307, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2.

[12] N. Doshi. Indexing data into Splunk remotely. 2010. URL http://blogs.splunk.com/2010/02/10/indexing-data-into-splunk-remotely/. Blog post. Retrieved March 16, 2011.

[13] Ericsson AB. *Erlang – Distributed Applications*. URL http://www.erlang.org/doc/design_principles/distributed_applications.html. Technical Documentation. Retrieved April 25, 2011.

[14] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66 – 73, July-Sept 2004. ISSN 1536-1268.

[15] G. Huerta-Canepa and D. Lee. An adaptable application offloading scheme based on application behavior. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 387 –392, March 2008.

[16] 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*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0155-8.

[17] I. Kelé andnyi, A. Ludanyi, J. Nurminen, and I. Puustinen. Energy-efficient mobile bittorrent with broadband router hosted proxies. In *Wireless and Mobile Networking Conference (WMNC), 2010 Third Joint IFIP*, pages 1 –6, 2010.

[18] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a computation offloading framework for smartphones. MobiCASE, 2010.

[19] L. Kleinrock. Nomadic computing–an opportunity. *SIGCOMM Comput. Commun. Rev.*, 25:36–40, January 1995. ISSN 0146-4833. URL http://doi.acm.org/10.1145/205447.205450.

[20] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51 –56, april 2010. ISSN 0018-9162.

[21] Microsot Corporation. Windows embedded automotive 7 solutions. URL http://www.microsoft.com/windowsembedded/en-us/evaluate/windows-embedded-automotive-7.aspx. Marketing Material Web Site. Retrieved April 26, 2011.

[22] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 46–57, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-096-8.

[23] D. Polberger. Component technology in an embedded system. Master's thesis, January 2010. URL http://www.polberger.se/components/thesis.pdf. Retrieved March 24, 2011.

[24] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 255–270, New

York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5.

[25] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10 –17, Aug. 2001. ISSN 1070-9916.

[26] J. a. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*, WICSA 3, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V. ISBN 1-4020-7176-0.

[27] V. Stirbu. A RESTful architecture for adaptive and multi-device application sharing. In *Proceedings of the First International Workshop on RESTful Design*, WS-REST '10, pages 62–65, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-959-6.

[28] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991. URL `http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html`.

[29] X. Zhang, S. Jeong, A. Kunjithapatham, and S. Gibbs. Towards an elastic application model for augmenting computing capabilities of mobile platforms. In *Mobile Wireless Middleware, Operating Systems, and Applications - Third International Conference, Mobilware 2010*, pages 161–174, 2010.