Paving the Way for Security in Cloud-Based Mobile Augmentation Systems

Andreas Reiter, Thomas Zefferer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
{andreas.reiter, thomas.zefferer}@iaik.tugraz.at

***Abstract*—Despite their steadily increasing capabilities, mobile end-user devices such as smartphones often suffer from reduced processing and storage resources. Cloud-based mobile augmentation (CMA) has recently emerged as a potential solution to this problem. CMA combines concepts of cloud computing and surrogate computing in order to offload resource-intensive tasks to external resources. During the past years, different CMA frameworks have been introduced that enable the development and usage of CMA-based applications. Unfortunately, these frameworks have usually not been designed with security in mind but instead mainly focus on efficient offloading and reintegration mechanisms. Hence, reliance on CMA concepts in security-critical fields of application is currently not advisable. To address this problem, this paper surveys currently available CMA frameworks and assesses their suitability and applicability in security-critical fields of application. For this purpose, relevant security requirements are identified and mapped to the surveyed CMA frameworks. Results obtained from this assessment show that none of the surveyed CMA framework is currently able to meet all relevant security requirements. By identifying security limitations of currently available CMA frameworks, this paper represents a first important step towards development of a secure CMA framework and hence paves the way for a use of CMA-based applications in security-critical fields of application.**

I. Introduction

The two computing paradigms cloud computing and mobile computing have dominated innovations and developments in the computing area during the past years. While both paradigms already show various advantages from a dissociated perspective, their full potential is best utilized when both mobile and cloud computing are combined and used together. This approach is followed by mobile cloud solutions, which employ the mobile character of end-user devices and virtually unlimited storage and processing resources of cloud solutions. It is hence unsurprising that mobile cloud solutions have attracted attention and gained popularity during the past years. Popular examples for such solutions are cloud-based image-processing applications, voice-recognition solutions, or navigation systems. All these solutions are available on current mobile end-user devices such as smartphones or tablet computers and outsource complex storage and processing tasks to cloud infrastructures in order to save local capacities on the mobile device.

Despite their continuously growing popularity, mobile cloud solutions still suffer from one conceptual drawback: most of them make use of dedicated cloud resources. If these resources become unavailable for some reason, the entire solution fails. For instance, mobile navigation systems do not work, if required map data cannot be downloaded in time from a predefined server. In situations, in which the mobile device is disconnected from the Internet, mobile navigation systems are hence usually useless. Similar limitations also apply to other mobile cloud solutions that dependent on a working Internet connection.

To overcome this limitation, the concept of cloud-based mobile augmentation (CMA) appears to be a promising approach. CMA-related concepts have been proposed recently e.g. by Chun et al. [1], Kosta et al. [2], Cuervo et al. [3], Rellermeyer et al. [4], or Dou [5]. These concepts enhance the basic idea behind mobile cloud solutions by breaking the dedicated link between the mobile application and its external cloud resources. Instead of relying on one or more predefined cloud resources, CMA-based applications make use of those processing and storage resources that are available just at the time of execution. For instance, they outsource processing tasks to other proximate end-user devices that are currently in reach. If available, CMA-based applications also make use of classical cloud resources. However, in contrast to classical mobile cloud applications, cloud resources are not the only outsourcing alternative for CMA-based applications. Thus, these applications show a higher degree of flexibility regarding external resources. This makes them more applicable and less vulnerable to failures in offline scenarios compared to mobile cloud solutions.

The dynamic outsourcing of tasks to available local and remote resources is a complex task. In most cases, required functionality needs to be integrated into and provided to CMA-based applications during their development and also at runtime. During the past few years, several CMA frameworks have been introduced that enable and facilitate the development and operation of CMA-based applications. Popular examples for currently available CMA frameworks are MAUI [3], CloneCloud [1], and ThinkAir [2]. So far, the main aim of these frameworks is the identification of and the reliable connection to available local and remote resources, as well as the provision of efficient and easy-to-integrate outsourcing mechanisms. Other aspects usually play only a subordinate role in current CMA frameworks so far. This especially applies to security aspects and to the protection of outsourced data. However, security is a crucial aspect for cloud-based solutions. This has been discussed by Rong et al. [6] and by Ryan [7] in detail. This also holds true for CMA-based applications, for which external resources are not predetermined. As secu-

rity considerations play only a minor role in current CMA frameworks so far, it is yet not clear if these frameworks are able to meet potentially high security requirements of CMA-based applications. Development and operation of CMA-based applications in security-critical fields of application is hence currently not advisable.

In this paper, we address this issue by assessing current CMA frameworks in terms of their capability to meet potential security requirements of CMA-based applications. For this purpose, we identify relevant security requirements of CMA-based applications that process security-critical data. We then survey current popular frameworks for the development and operation of CMA-based applications. Surveyed frameworks are subsequently classified according to established categorization schemes. Finally, we evaluate the surveyed CMA frameworks against the identified security requirements in order to identify those frameworks that are suitable for the development and operation of CMA-based applications that process security-critical data. This way, this paper contributes to a future application of the CMA approach also in security-critical fields of application.

II. Background and Related Work

The concept of CMA aims to enhance the computing power and storage capacity of mobile end-user devices by dynamically outsourcing resource-intensive tasks to external resources. The augmentation of mobile devices with the help of external resources is actually not a new idea. A similar approach has already been introduced in 2001 by Satyanarayanan [8] and has become known under the terms *cyber foraging* and *surrogate computing*. These terms cover technologies that enhance the computational power of resource-constrained devices with available resources from proximate devices such as laptops, desktop PCs or other mobile device, in order to enable the execution of applications requiring more computational power.

CMA develops the basic concepts of surrogate computing and cyber foraging further by incorporating the concept of cloud computing. Today, various applications already rely on the concepts of cloud computing e.g. to offload voice recognition, image processing, and other resource-intensive tasks to the cloud. Furthermore, cloud services are used to virtually extend the storage capacity of mobile devices. As a pre-condition, these techniques require a fast and reliable connection to the cloud service or, more generally speaking, to the Internet. The separation between locally and remotely executed tasks is defined statically during the development phase of the application. For classical cloud solutions, there are only very limited ways to dynamically consider factors like available bandwidth or latency. This limitation of classical cloud computing is overcome by CMA, which can be regarded as a combination of surrogate computing and cloud computing. CMA-based applications rely on the basic concept of surrogate computing and dynamically assign tasks to available external resources. In contrast to the plain surrogate-computing concept, outsourcing mechanisms of CMA-based applications are however not limited to proximate devices but can also rely on available cloud resources.

In this section, we provide an overview of relevant background information and related work on CMA. We start with a brief overview of the evolution of CMA and introduce in more detail the underlying concepts of cloud computing and surrogate computing. We then elaborate on different types of external resources that can be used by CMA-based applications. Finally, we discuss available offloading and reintegration mechanisms that are used by current CMA frameworks.

A. CMA Concepts

CMA basically combines the concepts of cloud computing and surrogate computing. These concepts are described in the following in more detail.

1) Cloud Computing: Cloud computing is a service-oriented approach to offer IT services to users based on a pay-as-you-go model. Cloud computing services are typically provided on three levels: *Infrastructure-as-a-Service (IaaS)* offers virtualized hardware platforms, *Platform-as-a-Service (PaaS)* offers ready-to-use runtime environments for own service development and *Software-as-a-Service (SaaS)* offers ready-to-use services with a particular purpose. One key advantage of cloud computing over traditional IT infrastructures is that service providers can offer their services without requiring high investments in advance, resulting in a reduced time to market. Furthermore, due to the dynamic nature of cloud computing and cloud resources, service providers can absorb load spikes, which in turn improves the experience for users.

The possibility to make use of virtually unlimited processing power and storage capacities makes cloud computing especially attractive for mobile end-user devices, which often suffer from limited capabilities. The use of cloud computing on mobile devices has become known under the term mobile cloud computing (MCC). Shiraz et al. [9] define MCC as a new paradigm in the field of utility computing, which extends the concepts to devices with constrained resources. To offload resource-intensive tasks, MCC utilizes different components, as illustrated in Figure 1. MCC does not only cover the outsourcing of computational power, but also e.g. the outsourcing of storage space among different devices.

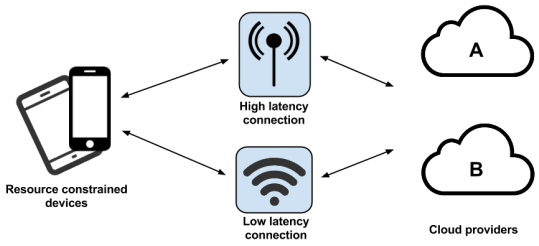

Fig. 1. Mobile Cloud Computing components

2) Surrogate Computing: Surrogate computing [8] also targets the augmentation of resource-constrained devices. In contrast to MCC, this concept considerably increases the set of utilized devices and also takes proximate resources such as notebooks, desktop computers, or other mobile devices into account. This way, lower latencies can be achieved for outsourcing processes compared to pure cloud-computing solutions.

This is relevant as according to Satyanarayanan et al. [10], even relatively low latencies of 33ms already significantly impact user experience. Satyanarayanan et al. conducted a test, in which an application was run locally and with different remote desktop applications utilizing wide area networks (WANs). In contrast to the local execution, either the frame rate of the remotely executed application dropped significantly or the application responded slowly. One way out would be to improve the latencies of WANs. However, recent developments rather go the way of increasing bandwidth than reducing latency. Therefore, it is not foreseeable that significant improvements will be made on the latency issue in the near future. It is hence reasonable to rely on approaches such as surrogate computing that provide outsourcing with low latencies.

Figure 2 shows the extended component model for surrogate-computing systems. Resource-constrained devices can offload tasks to other surrounding devices, with the limitation that these devices should have a similar movement pattern so that they remain in range for a long time. Devices could also use so-called Cloudlets for computational offloading, or as a gateway to other cloud resources. The concept of Cloudlets has been introduced by Satyanarayanan et al. [10] and describes resources that, from the user's perspective, provide computational resources and are hence comparable to classical cloud resources.

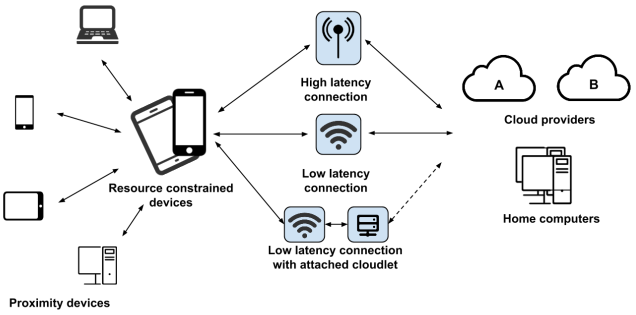

Fig. 2. Components of a Cyber Foraging system

B. CMA Resources

Cloud computing and surrogate computing form the basis for CMA. Hence, CMA-based applications make use of both cloud resources and resources that are provided by proximate devices and can be accessed through surrogate-computing approaches. Abolfazli et al. [11] introduce a classification of utilized resources of CMA-based applications. They differentiate between *Distant Immobile Clouds*, *Proximate Immobile Computing Entities* and *Proximate Mobile Computing Entities*. These different types of resources are introduced in the following in more detail. In practice, CMA-based applications should follow a hybrid approach and use more than one type of resources. A comparison of the different types of resources is also provided in Figure 3.

1) Distant Immobile Clouds: Distant immobile clouds are classical cloud resources from major cloud-service providers with large data centers. Additionally, also private-cloud resources from companies and organizations can be classified as distant immobile clouds. The connection to these resources

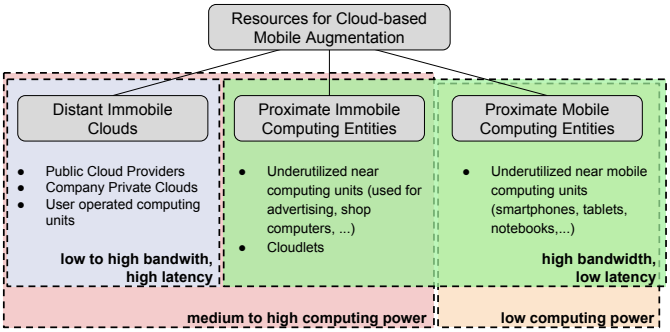

Fig. 3. Overview of resources utilized by CMA-based applications

is established through the Internet with measures in place to maintain connection security. Large data centers are highly efficient in terms of power consumption and utilization of available resources. However, due to their large distance to the users, the latency is relatively high compared to other types of resources. This can significantly impact the user experience and can potentially lower the performance of the whole application. If only low bandwidth and high-latency connections to the cloud resources are available, distant immobile clouds are typically not the first choice for CMA-based applications to offload tasks.

2) Proximate Immobile Computing Entities: Proximate immobile computing entities are resources that are located in the surrounding of users, e.g. in public places. Such resources may be used to show advertisements, play music, or are used as simple input devices. It is likely that they have a fast wired or wireless connection to the Internet, and are, at least most of the time, not fully utilized. These resources can also be used by surrounding users and their CMA-based applications to offload tasks.

Another approach in this category is based on so-called Cloudlets, which have been introduced by Satyanarayanan et al. [10]. The concept of Cloudlets is sometimes also referred to as Cloud-in-the-Box. Cloudlets are a special kind of proximate immobile computing entities. They are dedicated to CMA-based applications that need to offload resource-intensive tasks. Cloudlets can also be used as low-latency gateways to other offloading resources such as distant immobile clouds. A known challenge of the concept of Cloudlets is the trust relationship between the user and the Cloudlet provider. Without further security measures, the Cloudlet provider can theoretically spy on the user and can extract sensitive or personal information.

3) Proximate Mobile Computing Entities: Proximate mobile computing entities refer to the utilization of surrounding mobile devices like smartphones, tablet computers, or notebooks. Because mobile devices are dedicated to be on the move, using proximate mobile computing entities is only feasible, if they have a similar movement pattern than the user. Only in this case offloading targets will be available during a sufficient period of time. Security concerns for proximate immobile computing entities also apply to proximate mobile computing entities, as used resources are under control of another person and can also be infected with malware.

C. CMA Building Blocks

CMA-based applications need to integrate several basic building blocks that implement required functionality. Most relevant functionality is covered by the so-called *offloading and reintegration mechanism*, which is a core functionality of each CMA-based application. The offloading and reintegration mechanism is composed of two components: the *partitioning algorithm* and the *offloading component*.

The partitioning algorithm separates an application into parts that can be run remotely and into parts that are required to run locally. A local execution is for instance required for parts that require an interaction with the local user interface or with sensors of the mobile device such as GPS, compass, or camera. The partitioning algorithm also requires an estimation for the efficiency of offloading under current conditions, in order to achieve a suitable partitioning. The partitioning itself can happen statically at compile time, dynamically at runtime, or can be a combination of both. In practice, the developer usually defines a static pre-partitioning of the application by marking specific classes and methods. Alternatively, the offloading can also be based on separate threads. Additionally, a runtime component analyzes, which parts are candidates for offloading under current conditions and takes into account parameters such as available bandwidth, latency, energy consumption, or available computational power.

The offloading component is the second basic building block that needs to be implemented by CMA-based applications. This component outsources parts of the application to available remote resources. The outsourcing of components can be based on different concepts with different levels of granularity. These concepts are explained in the following in more detail.

1) Virtual Machine Based Approaches: Virtual machine (VM) based approaches, as described by Chun et al. [1], aim to provide a runtime environment, which is similar and compatible to the environment of the resource-constrained device. Following this approach, the VM can execute unchanged mobile-device applications, lowering the required development effort and enabling a fast and broad rollout. The disadvantages of this approach are obvious. First, a vast amount of different virtual-machine images reflecting different hardware and operating-system combinations is required. Second, for efficiency reasons, a VM will only be started when required, which results in a few seconds delay for the user. Third, all applications need to be available at the resource provider's side or need to be uploaded, which again results in a delay. Finally, the VMs consume a non-negligible amount of resources, which may even exceed the resources required by the application. In general, VM-based approaches are hence inappropriate in most cases.

2) Class or Method Based Concepts: As described by Cuervo et al. [3], class or method based concepts provide a more fine-grained approach and focus on the offloading of classes or methods instead of entire applications. Before methods are invoked, it is decided, if the method is executed locally or remotely. Depending on the utilized programming language, it may not be required to provide a compatible execution environment. Using managed programming languages (for example the Java programming language or the .NET environment), it is possible to transfer the current state of an application to another machine, which may be running a different operating system. Class or method based concepts require more support from the developer, but in return eliminate delays induced by the VM-based approach. These concepts might also require the mobile device to upload parts of the application to the offloading target. However, by using smart extraction of the relevant application parts, it is possible to minimize the size of these data.

3) Object-Based Concepts: Object-based concepts as proposed by Sinha et al. [12] extend the class-based approach to a even more-fine grained level. Sinha et al. [12] introduce object-based concepts by means of the concrete example of an image-manipulation software, where a certain class is used to apply different effects. With a class-based approach, this class would always be offloaded to the same target, or would always run locally. Utilizing the object based approach, the system bases the offloading decision on the prediction of the required computational effort for one specific instance of a class, i.e. an object. This way, different instances of the same class can be offloaded to different targets.

III. Requirements

Independent of the utilized external resources and the followed outsourcing approach, the main aim of current CMA frameworks is the provision of efficient offloading mechanisms. Security aspects usually play only a marginal role in these frameworks. This is problematic, as a lack of security renders the use of CMA-based applications in security-critical fields of application impossible. As a first step towards a solution to this problem, this section identifies requirements for security-critical CMA-based applications. The list of requirements will be used in Section V to evaluate and assess current CMA frameworks.

Requirements listed in this section have been derived from related literature. Some requirements are very similar to the requirements of general cloud computing, others can be derived from requirements of classical IT systems. According to Ryan [7], cloud computing differentiates from the paradigms of classical IT systems in the following key points:

- Cloud resources are shared among a certain amount of users. Every user is a potential attacker.
- Cloud resources could be retrieved using unsafe protocols, or could be transferred using public networks.
- The cloud provider has full responsibility of the data integrity of all users.
- The cloud provider and its sub contractors potentially have access to all stored data.

These key points influence cloud computing related security challenges as defined by Rong et al. [6]:

Location of data: In the field of cloud computing, big players are acting around the globe with data centers in regions that are beneficial in terms of location or costs of operation. Users usually do not have control or tracing possibilities where cloud providers store or backup data. Cloud providers will store data at those locations that are most attractive for them.

Multi-tenancy: Generally, cloud services are built to serve multiple users and user groups. As a user, one needs to trust the cloud provider to have mechanisms in place to isolate different users and their data from each other.

Monitoring and logging: With the increasing amount of applications being migrated to the cloud, it is of highest importance for the service operator to have decent logging and monitoring mechanisms in place. This goes hand in hand with the requirement for multi-tenancy, as the logging needs a complete isolation between different services and users. This is mandatory as log files may contain sensitive data.

Cloud-related standardizations: For specific cloud-related branches, standardizations are available today. Generally, a provider only supports applications which are developed particularly for one specific platform (vendor-lock-in).

From the identified security challenges of cloud computing, security requirements can be derived. The derivation and identification of security requirements for cloud-based solutions has been a topic of scientific interest for several years. Accordingly, several publications are available, which focus on security requirements for cloud computing in general [13] [14]. To a certain extend, these requirements also apply to CMA-based applications. CMA can be regarded as specific framework or application for cloud-computing resources. CMA approaches in general are not bound to classical cloud resources, but can also utilize other types of resources including proximate devices. From a devices perspective, these resources can also be seen as just another special type of cloud resources. Due to the similarity between cloud computing and CMA, some of the cloud computing related security requirements as defined by Honer [13] and Iankoulova [14] also apply to CMA systems without changes. In summary, the following requirements can be regarded as relevant for security-critical CMA-based applications.

- ***R1 - Reliability:*** Reliability refers to the availability of the system. The reliability is negatively affected by the presence of single points of failures in the system. An example is a central server, which is organizing the whole offloading process.
- ***R2 - Integrity:*** Integrity means that data remains consistence over its whole life cycle. In the context of CMA, integrity means that cloud resources execute offloaded code without applying any modifications. As illustrated in Figure 4, the cloud provider could modify the offloaded code unnoticed by the user. The modified code could then introduce a different behavior or return erroneous results. As modifications are applied remote from the user and its CMA-based application, they are hardly detectable.

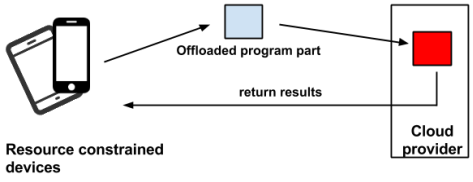

Fig. 4. Integrity requirement

- ***R3 - Privacy and confidentiality:*** This requirement tackles the issue of unauthorized third parties trying to access sensitive information. In a CMA context as seen in Figure 5, the cloud providers generally do not deal with data as, for example, a storage provider does. However, they execute parts of an application that potentially contains sensitive information. It might be more complex to extract relevant data, but the data potentially also contains highly relevant information like passwords, user names and others.
- ***R4 - Non-repudiation:*** Non-repudiation means that a party cannot deny that she has sent a message. Only the sender is in charge of the necessary information to produce messages that refer to the particular party. In the CMA context it is important that resources can link requests to particular clients without a doubt as resources may expose sensitive information to the clients and resources may charge the clients based on the consumed resources. Therefore, it is of high interest for attackers to impersonate other clients.
- ***R5 - High isolation level:*** This requirement is tightly coupled with Requirement R3, but due to the discussed types of offloading mechanisms is a very important requirement for CMA frameworks. Depending on the used mechanism, the system has a low to high isolation level between different users. On a classical virtual machine based approach, where each offloaded application gets its dedicated virtual machine, a high isolation level can be achieved. A high isolation level implies a low risk of outbreak, as the attacker would need to find a security flaw in the virtual-machine hypervisor. On the other end, there are systems that better utilize the available resources, but in contrast may share a single virtual machine and execution environment among different users and applications. Without further security measures, it seems much easier to get access to data of other applications running in the same environment.
- ***R6 - Misuse of provided resources:*** All requirements identified so far relate to the user and his or her data. Considering a CMA-based application, providers of external resources generally have less control of the executed application and therefore also need to be protected. This requirement addresses ways for resource providers to assure that users do not misuse provided computational power for fraudulent activities.

The six security requirements defined in this section will be used to systematically assess security capabilities of current CMA frameworks. Relevant frameworks that will be assessed are surveyed and briefly sketched in the next section.

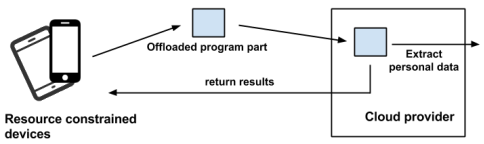

Fig. 5. Privacy and confidentiality requirement

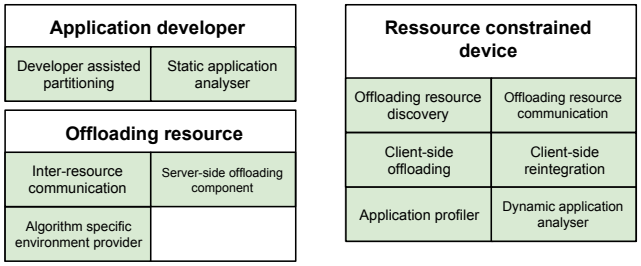

Fig. 6. Offloading system components

IV. Existing CMA Frameworks

In this section, an overview of existing CMA frameworks is provided in order to sketch the current state of the art. In general, CMA frameworks comprise components that assist developers in creating CMA-based applications, components that enable the use of CMA-applications on resource-constrained devices, and components that are part of external offloading resources used by CMA-based applications. This is illustrated in Figure 6, which identifies and lists relevant components of CMA frameworks.

During the development process of a CMA-based application, the application developer can assist in the pre-partitioning process as described in Section IV-C and illustrated in Figure 6 by the developer assisted partitioning component. For this purpose, the developer can use static application analyzer tools provided by the CMA framework. These tools help the developer to identify those program parts that need to run locally and those that can be executed remotely.

The resource-constrained mobile device may contain a dynamic analysis component, which operates on data from the application profiler and from the offloading resource discovery component. The application profiler monitors the execution of the application and provides time or memory consumption estimations for particular parts of the program. The offloading resource discovery component has connections to available external resources and can provide computational power on demand. The communication with offloading resources is performed using the offloading resource communication component, which may use simple web APIs or more complex systems like peer-to-peer networks. The central component, which is split among the mobile device and the offloading resource, consists of the client-side and server-side offloading and reintegration components. These components perform the migration of the execution to the remote resource and reintegrate the results once finished.

The server-side offloading component runs the offloaded part on the available resource and sends back the results to the calling client. The offloading resources may also have mechanisms for inter-resource communication and may distribute work among other resources. Depending on the utilized offloading method, an algorithm-specific environment provider component is required to provide the execution or virtual machine environment to the offloading component.

Figure 6 shows that CMA frameworks can potentially comprise various different components that enable the use of CMA concepts. In practice, CMA frameworks typically support and provide only a subset of all possible components. In the following subsections, relevant existing CMA frameworks are introduced. Focus is put on those frameworks that do not limit the applications to particular cases, but are generally applicable in different fields of application.

A. AlfredO

AlfredO [4] is based on the R-OSGi [15] middleware, which improves the concepts of OSGi [16] to a distributed manner. OSGi enables Java application developers to loosely couple modules in an efficient way. R-OSGi extends this concept and enables distribution of modules among different computing units. The vision of AlfredO is to use mobile phones as generic interfaces to proximate devices. Additionally, AlfredO preserves the device's unique features like touch screen or different sensors. To maintain the security of the devices and to achieve the defined goals, AlfredO introduces three main paradigms:

- AlfredO utilizes a service-oriented software distribution to execute software using the R-OSGi framework and to retrieve remote interfaces.
- A multi-tier service architecture is defined, consisting of presentation tier, logic tier, and data tier. The execution can be distributed dynamically among mobile devices and offloading resources within the boundaries of the defined tiers.
- The device-independent presentation tier utilizes advantages of all platforms. The framework only sends out a high-level description of the user interface. The end-user device then generates a corresponding user interface utilizing available technologies.

AlfredO has a very small footprint on mobile devices with a minimal size of 290 kBytes. In practice, the size heavily depends on the provided components, e.g. different renderers. As the framework is based on R-OSGi and OSGi, it requires mobile platforms, on which Java virtual machines are available, and which additionally support the concepts of R-OSGi. As Java in general is available on various devices [17], the former does not raise any problems. However, the Java VMs are only available in a very limited version. Hence, each platform needs a re-evaluation in terms of R-OSGi compatibility. Furthermore, AlfredO requires the developer to provide a static partitioning of the application. Therefore, a fine-grained dynamic partitioning is not possible with AlfredO.

B. Misco

Misco [5] utilizes the *Map-Reduce* mechanism to complete tasks. This mechanism splits tasks into smaller ones, and processes them in parallel. The processing takes place in two steps: The first step is called map phase. There, the processing function is applied to a set of input data, which results in intermediate results. The second step, i.e. the reduce phase, groups the intermediate results to get the final result.

The Misco environment consists of a server component, which enables the user to define new tasks via a web interface, and multiple workers, which process these tasks. Each Misco worker processes a single map or reduce task at any given

time. Misco workers could for instance also be other mobile devices.

The concept of Misco differs from other described frameworks in the way how tasks are added and managed. The critical path in the system is the central Misco server, which manages and coordinates all of the available resources. If this component is taken down, the whole system fails. Dou [5] introduced a prototype system that is based on the Python programming language and offers a wide compatibility with existing systems. This prototype shows the feasibility of the underlying concept of Misco. However, beside the fact that Python does not really have an impact on mobile devices, it is a potentially challenging and time-consuming task to port existing applications to this framework.

C. MAUI

MAUI has been introduced by Cuervo et al. [3]. Its main goal is to be as energy and battery saving as possible. Conceptually, MAUI is limited to managed code environments. The prototype introduced by Cuervo et al. [3] is for instance limited to Microsoft .NET [18]. MAUI uses a method-level offloading approach and hence provides fine-grained outsourcing opportunities. Methods that can be offloaded are marked as remotable. An optimization framework is utilized to decide if a method should be offloaded under current conditions such as available connectivity, round-trip-time, or effort required to transfer the current state of the application. There, the developer decision on remotable methods is only an initial partitioning, which gets fine-tuned during runtime.

MAUI mainly focuses on the offloading part and not on details on how to manage or combine available resources. If the connection to the offloading server is lost, the framework restarts the process locally, resulting in a short delay only. MAUI utilizes a profiling and solver model, where all executions are supervised by the profiler and changes are recorded. The recorded data is used as input for the solver to decide which methods to offload.

The most interesting aspect of MAUI is its simplicity. No prior knowledge is required to make use of this CMA framework. Only few internals need to be known by the developer to improve the execution of applications using MAUI. Although MAUI focuses on the .NET programming language, it can easily be adopted to other programming languages. Unfortunately, the .NET framework is still not very popular on mobile devices and basically restricted to the Windows Phone platform. According to an IDC study from 2014 [19] only 2.5% of all smart phones are currently equipped with Windows Phone. Although there are ways to execute .NET applications also on other platforms, it is not very popular at the moment and may not be of interest for platform vendors. Even if the MAUI system gets ported to other programming languages, compatibility issues would still remain.

D. CloneCloud

The CloneClouds [1] concept aims to offload tasks to platform clones in the cloud with the goal to optimize the execution time and to minimize the energy consumption on the mobile device. The cloned virtual machines in the cloud need to be as similar to the mobile device as possible. In contrast to MAUI, CloneCloud can operate on unchanged applications, utilizing a mixture of static and dynamic partitioning algorithms.

The static analysis stores migration and reintegration points for later usages. They need to fulfill three fundamental properties. First, they must not contain device-specific functionality. Second, methods accessing native states always need to be executed on the same target (device or clone). Third, nested offloading operations are not permitted. The dynamic analysis calculates a cost estimation of the identified program parts using different execution parameters. Based on the collected data, the solver decides at runtime if a method is offloaded.

The prototype introduced by Chun et al. [1] is based on Java and Android. Therefore, it is theoretically able to operate on every available application. The migration and reintegration points are induced using special Java-VM commands. Therefore, for the framework to run, it is necessary to provide a modified Android version. No modified executable will run on unmodified versions of Android. As the distributions of currently used versions of Android is quite scattered from version 2.2 to 4.4 [20], it is unlikely that this technology will heavily be used in practice.

E. ThinkAir

ThinkAir, which has been introduced by Kosta et al. [2], combines the concepts of MAUI and CloneCloud to add more scalability. ThinkAir can dynamically allocate resources by starting and stopping virtual machines. ThinkAir has been designed based on four major principles:

- *Dynamic adoption to changing environments:* As CMA frameworks mainly target resource-constrained mobile devices, it is key that frameworks dynamically adapt to changes in their environment.
- *Ease of use for developers:* To achieve a broad usage and acceptance, the framework must be usable by both inexperienced and experienced users.
- *Performance improvement through cloud computing:* The main goal of ThinkAir is to improve the performance of applications and to minimize the energy consumption.
- *Dynamic scaling of computational power:* The framework should be applicable for applications with different requirements.

ThinkAir provides a pre-partitioning mechanism, similar to the approach used by MAUI and introduces an additional compilation step, which generates code for the remote execution. The prototype introduced by Kosta et al. [2] has been developed for Java and has been based on Android. It eliminates the limitations induced from CloneCloud by combining the approaches from MAUI and CloneCloud. However, it does not eliminate the induced overhead due to the provisioning of virtual machines as described in Section IV-D.

F. Cuckoo

Cuckoo, which has been introduced by Kemp et al. [21], is a CMA framework based on Android with slightly different

goals compared to CloneCloud or ThinkAir. The top goal of Cuckoo is to achieve a seamless integration in the development workflow and in integrated development environments to foster the ease of use. Furthermore, Cuckoo aims for simplicity on the resource providers' side. It does not require complete virtual machines. Instead, it works with Java VMs, which can be provided without much effort.

Cuckoo does not require the same code to be executed locally and remotely. The integrated development environment generates interfaces for the remotable parts. The developer can choose to use the same implementation of remotable parts for local and remote execution or provide different implementations depending on the location of execution. In fact, during compilation the resulting application always contains a separate library containing the parts that may be executed remotely.

In genereal, this approach seems more efficient than executing the same implementations on all platforms without the possibility to optimize it for certain platforms. On the other hand, Cuckoo does not fulfill the requirement of an easy migration of applications to the cloud.

G. COSMOS

COSMOS has been introduced by Shi et al. [22] and aims to provide *Offloading-as-a-Service* in order to minimize offloading costs. COSMOS is also an VM-based approach for Android. Other systems like CloneCloud always dedicate a VM to one specific user. COSMOS tackles this issue by introducing a COSMOS Master, which supervises all available resources and can early react to prevent spikes by starting new virtual machines. Moreover, COSMOS shares VMs among multiple users.

The offloading algorithm has not been described in detail in [22]. Anyhow, the described concepts eliminate the previously described delay for starting new resources due to the COSMOS Master and focus on cost minimization due to a smart control of the resources.

H. Offloading for Web applications

Hwang et al. [23] propose a framework to offload web applications based on the HTML5 Web Worker specification [24]. Web workers are comparable to separate threads in other programming languages but operate under a higher isolation level. They can only communicate with the main thread using the *PostMessage* mechanism. Hwang et al. [23] propose to offload the separate worker threads using HTML5 Web Sockets and reintegrate the results once processing has finished. The technique is the first one that provides offloading capabilities to the emerging field of web-based applications.

V. Evaluation

The conducted survey has revealed that there is currently a heterogeneous ecosystem of different CMA frameworks. These frameworks differ significantly in terms of provided features and underlying concepts. In this section, all surveyed frameworks are evaluated systematically in order to enable direct comparisons. The evaluation comprises two steps. First, all surveyed CMA frameworks are classified according to their supported external resources and their applied outsourcing approach. Subsequently, the provided security of all frameworks is assessed with the help of the security requirements identified in Section III.

TABLE I. Overview of utilized resources per framework

	Distant Immobile Clouds	Proximate Immobile Computing Entities	Proximate Mobile Computing Entities	VM-based approach	Class- or method-based approach	Object-based approach
AlfredO	·	·	✓	·	✓	·
Misco	·	·	✓	·	✓	·
MAUI	✓	·	·	·	✓	·
CloneCloud	✓	·	·	✓	·	·
ThinkAir	✓	·	·	✓	·	·
Cuckoo	✓	·	·	·	✓	·
COSMOS	✓	·	·	✓	·	·
Web	✓	·	·	·	·	·

A. Categorization

Depending on their underlying concept, CMA frameworks make use of different external resources. Concretely, they can rely on distant immobile clouds, proximate immobile entities, or proximate mobile computing entities. Furthermore, CMA frameworks follow different approaches to outsource tasks. Possible approaches are outsourcing based on VMs, class or method based concepts, and object-based concepts. Table I shows, which types of external resources and outsourcing approaches are used by the surveyed CMA frameworks.

The results in Table I show, that none of the evaluated frameworks utilizes multiple classes of resources, each framework sticks to one specific class. Proximate immobile computing entities are not considered at all. A majority of the evaluated systems only utilize distant immobile clouds in the form of public-cloud resources. Furthermore, on the utilized offloading methods, AlfredO and Misco have an outstanding position. AlfredO is mainly designed as a universal smart remote control for applications where the user interface dynamically adapts to the device. Misco utilizes a map-reduce approach where worker threads distribute all over different mobile devices, and is not designed to dynamically partition applications. The other frameworks distribute among the VM-based and class- or method-based approaches. However, the full flexibility and performance gain of class or method based offloading can only be employed when combined with multiple different types of resources types. Interestingly, the object-based approach is not utilized by any of the frameworks. This remains rather a theoretical concept, as its implementation requires much more effort.

B. Security Assessment

This section assesses security capabilities of all surveyed CMA frameworks by means of the security requirements

defined in Section III. Considering pure cloud solutions, most identified security requirements can be fulfilled using available technologies, if the cloud provider is only used as a storage provider. In this case, reliability can be guaranteed by using multiple cloud providers and by replicating content. Integrity can be assured by additionally storing hash values or digital signatures. Privacy and confidentiality can be preserved by applying strong encryption and by limiting the storage of data to trusted entities. The requirement for non-repudiation does not really apply to storage providers but is comparable with applying digital signatures to preserve integrity. The requirement for a high isolation level finally maps to the use of strong passwords and two-factor authentication.

In case the cloud service does not only store data but does also process it, the situation gets more complex. One promising approach is the use of fully homomorphic encryption, which enables operations on encrypted data. Other solutions require provision of the used encryption key to the cloud-service provider, which has an identical protection level as applying no encryption at all. Unfortunately, also CMA concepts require the processing of data by external cloud resources. Relevant security requirements can hence not be assumed to be fulfilled when CMA concepts are followed. The capabilities of current CMA frameworks to meet relevant security requirements are hence assessed in detail in the following subsections. A summary of this assessment is provided in Table II. The table contains three different symbols: "✓" for requirements which are completely fulfilled, "✗" for requirements which are not fulfilled and "∼" for requirements which are partly fulfilled or could be fulfilled based on the documentation. If table cells are left blank, relevant information is not available for the particular framework.

1) Reliability: Reliability can only be provided by systems that do not have a single point of failure or that feature a fallback mechanism. AlfredO is a distributed system and does not maintain any uplink to central servers. From a user's perspective, the reliability of AlfredO can be regarded as high. This is in stark contrast to Misco, where a central Misco server coordinates all workers. Without the Misco server, the workers can not proceed or deliver their results. The approaches followed by MAUI, CloneCloud, ThinkAir, and Cuckoo all focus on the offloading technique and development-workflow integration. They do not tackle the issue of reliability and assume that they are connected to their assigned resources. We do not consider the reliability requirement as not fulfilled for these frameworks, because the systems could be extended easily to provide more reliability. COSMOS further constrains and optimizes procedures by better utilizing available resources. A central server, i.e. a single point of failure, which manages the available resources, is introduced. The presented web application based offloading technique is rather a concept so far, and does not yet implement a complete system. We therefore cannot evaluate the requirement for reliability for the web application based offloading framework.

2) Integrity: The requirement for integrity refers to the execution of the correct offloaded code. In fact, users cannot verify if indeed the correct data is executed on the remote resource. None of the described frameworks tackles this issue. Only the Cuckoo framework has an exceptional position, as it enables the execution of different implementations depending on the execution location. One possible solution to mitigate the issue of integrity is to utilize multiple resources and execute the offloaded parts simultaneously on multiple sites. This might raise some other issues like side effects of multiple executions, but at least the issue of wrong response values is tackled. Still it does not prevent the cloud-service provider from modifying the offloaded code and executing code with side effects not reproducible by the user.

3) Privacy and Confidentiality: Privacy and confidentiality are closely related to the requirement for integrity as it also concerns the respectability of the executed code. In contrast to the requirement for integrity, it focuses rather on the data and sensitive information provided by the user. Again, none of the frameworks tackles this issue. The provider could inject code in the offloaded parts without modifying the intended functionality but extracting user-specific and sensitive information. The only suitable solution seems to be the establishment of trust relationships with providers and to only use resources from trustworthy entities for critical offloading operations.

4) Non-Repudiation: In a development environment, the requirement for non-repudiation is closely related to privacy and confidentiality. When it comes to productive services, non-repudiation is also of interest to resource providers, because they may charge the users based on their actual usage of resources. This issue is not directly tackled by any of the described frameworks, but seems achievable for systems, where resources are exclusively bound to particular users, or for systems, where a master is managing the available resources and is assigning users to them. This applies to MAUI, CloneCloud, ThinkAir, Cuckoo, or COSMOS. For the web application based offloading approach, insufficient information is available to decide on that issue. For AlfredO and Misco, this requirement seems hard to achieve, due to their distributed nature and their different goals compared to the other frameworks.

5) High Isolation Level: The assessment of the requirement for a high isolation level is based on the utilized offloading technology. We cannot provide a statement for AlfredO, because of the different goal of this framework. Similarly, we cannot provide an assessment for the web application based offloading approach, due to lack of relevant information. Although the proposed Misco framework does not utilize different virtual machines, a high isolation level is achievable due to the very clear worker separation and structure of the offloaded code. MAUI, in contrast, may use a single resource for multiple users and without further measures, it may be possible to access other processes on the same resource. CloneCloud, ThinkAir, and Cuckoo are all based on virtualization techniques, where a single machine is only used by a single tenant. Therefore, a high isolation level is provided by these frameworks. COSMOS extends the concepts of MAUI and CloneCloud and better utilizes the available resources by using the same resource for multiple users. This may lower the isolation level.

6) Misuse of Provided Resources: The former requirements all tackled issues affecting the users of a system. In contrast, this requirement concerns the security of the resource provider. The best approach to prevent misuse of provided resources is the analysis of offloaded parts. Although the issue is not concerned by any of the discussed frameworks, we

TABLE II. EVALUATION OF THE REQUIREMENTS

	AlfredO	Misco	MAUI	CloneCloud	ThinkAir	Cuckoo	COSMOS	Web
R1 - Reliability	✓	✗	∼	∼	∼	∼	✗	
R2 - Integrity	✗	✗	✗	✗	✗	∼	✗	✗
R3 - Privacy and confidentiality	✗	✗	✗	✗	✗	✗	✗	✗
R4 - Non-repudiation	✗	✗	∼	∼	∼	∼	∼	
R5 - High isolation level		✓	✗	✓	✓	✗	∼	
R6 - Prevent misuse of provided resources	✗	∼	∼	∼	∼	∼	∼	

consider systems exclusively utilizing mobile devices to not have enough resources to conduct such an analysis. For cloud-resource providers, it is easier to build a database of trusted offloaded parts.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have surveyed, classified, and assessed currently available CMA frameworks that enable development and usage of CMA-based applications. The conducted survey has revealed that current CMA frameworks differ significantly in terms of implemented offloading approaches and employed external resources. The conducted security assessment has shown that none of the surveyed frameworks is able to meet requirements of security-critical applications. We hence have to conclude that currently available CMA frameworks are not suitable and applicable for security-critical fields of application so far.

For the near future, we plan to further intensify our work on the application of CMA concepts in security-critical environments. Concretely, we plan to develop a CMA framework that meets relevant requirements of applications storing and processing security-critical data. The work and the results presented in this paper are a solid basis for these attempts. This way, this paper represents a first and important step and paves the way for a future use of CMA-based solutions in security-critical fields of application.

REFERENCES

[1] B. Chun, S. Ihm, and P. Maniatis, "Clonecloud: elastic execution between mobile device and cloud," *Proceedings of the sixth ...*, pp. 301–314, 2011. [Online]. Available: http://dl.acm.org/ft_gateway.cfm?id=1966473\&type=pdf

[2] S. Kosta, A. Aucinas, and R. Mortier, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," *2012 Proceedings IEEE INFOCOM*, pp. 945–953, Mar. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195845

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Stefan, R. Chandra, and B. Paramvir, "MAUI : Making Smartphones Last Longer with Code Offload," *Energy*, vol. 17, pp. 49–62, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1814441

[4] J. S. Rellermeyer, O. Riva, and G. Alonso, "AlfredO : An Architecture for Flexible Interaction with Electronic Devices," pp. 22–41, 2008.

[5] A. Dou, "Misco : A MapReduce Framework for Mobile Systems ," 2010.

[6] C. Rong, S. T. Nguyen, and M. G. Jaatun, "Beyond lightning: A survey on security challenges in cloud computing," *Computers & Electrical Engineering*, vol. 39, no. 1, pp. 47–54, Jan. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0045790612000870

[7] M. D. Ryan, "Cloud computing security: The scientific challenge, and a survey of solutions," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2263–2268, Sep. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212003378

[8] M. Satyanarayanan, "Pervasice Computing: Vision and Challenges," pp. 10–17, 2001.

[9] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," vol. 15, no. 3, pp. 1294–1313, 2013.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14–23, 2009.

[11] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 337–368, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6563280

[12] K. Sinha and M. Kulkarni, "Techniques for Fine-Grained, Multi-site Computation Offloading," *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 184–194, May 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5948609

[13] P. Höner, "Cloud Computing Security Requirements and Solutions : a Systematic Literature Review."

[14] I. Iankoulova, "Cloud Computing Security Requirements : a Systematic Review," 2011.

[15] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi : Distributed Applications Through Software Modularization," pp. 1–20, 2007.

[16] OSGi Alliance, *OSGi Service Platform, Core Specification, Release 4, Version 4.3*, 2011. [Online]. Available: http://www.osgi.org/Download/File?url=/download/r4v43/r4.core.pdf

[17] "Java Verified - Table of Supported Devices," 2014. [Online]. Available: http://javaverified.com/device_matrix

[18] Microsoft, "Overview of the .NET Framework." [Online]. Available: http://msdn.microsoft.com/en-us/library/zw4w595w\%28v=vs.110\%29.aspx

[19] IDC, "Smartphone OS Market Share, Q2 2014." [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[20] "Android Dashboards." [Online]. Available: https://developer.android.com/about/dashboards/index.html

[21] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," *Mobile Computing, Applications, ...*, pp. 59–79, 2012. [Online]. Available: http://www.springerlink.com/index/U6777405301NP063.pdf

[22] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "COSMOS : Computation Offloading as a Service for Mobile Devices," 2014.

[23] I. Hwang and J. Ham, "Cloud Offloading Method for Web Applications," *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 246–247, Apr. 2014.

[24] "Web Worker Specification." [Online]. Available: http://www.w3.org/TR/workers/