Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461 ZVR: 948166612 UID: ATU60778947

PLATTFORMUNABHÄNGIGES CLOUD-BASED MOBILE AUGMENTATION SYSTEM

VERSION 1.0 - 22. OKTOBER 2015

Andreas Reiter – andreas.reiter@a-sit.at

Zusammenfassung:
Durch die ständige Verfügbarkeit von mobilen Geräten halten diese unaufhaltsam Einzug in das tägliche Leben der Benutzer. Außerdem werden mobile Geräte immer leistungsfähiger, sind aber trotzdem stark eingeschränkt in Bezug auf deren Laufzeit wegen einer begrenzten Kapazität der verwendeten Akkus. Um sich diesem Problem anzunehmen, wurden unterschiedliche Systeme entwickelt um dynamisch externe Ressourcen zu allokieren und diese dem mobilen Gerät zur Verfügung zu stellen. Es stellt sich aber heraus, dass einerseits alle verfügbaren Systeme sehr stark auf ausgewählte Plattformen beschränkt sind und andererseits keines der vorhandenen Systeme sich Sicherheitsaspekten widmet.
In diesem Projekt wird eine neue Architektur vorgestellt, die es erlaubt, dynamisch externe Ressourcen zu allokieren und bereits auf Architekturebene die definierten sicherheitsrelevanten Anforderungen erfüllt. Die Proof-of-Concept Implementierung nimmt sich außerdem dem Problem der Interoperabilität an und ist auf allen gängigen Plattformen und Betriebssystemen lauffähig. Die anschließende Evaluierung zeigt die Vorteile und Performancegewinne des Frameworks auf.

Inhaltsverzeichnis

1. Einleitung

Trotz der ständigen Verfügbarkeit von mobilen Endgeräten und Verbesserungen in Performance und Leistungsfähigkeit, ist die Verwendbarkeit durch die Akkulaufzeit stark begrenzt. Werden mobile Geräte intensiv genutzt und mit rechenintensiven Aufgaben belastet, kann es durchaus notwendig sein, diese mehrfach täglich aufzuladen. Um dem entgegen zu wirken, wurden von Softwareherstellern verschiedene Konzepte entwickelt.
Ein klassischer Ansatz ist, rechenintensive Aufgaben in die Cloud auszulagern und als API, Applikationen zur Verfügung zu stellen. Nachteil dieses Ansatzes ist, dass das Endgerät eine ständige Verbindung zum jeweiligen Cloud Provider benötigt, und sehr abhängig von der Qualität (Latenz, verfügbare Bandbreite) der Internetanbindung ist.
Deshalb wurden Ansätze entwickelt, die entweder dynamisch Ressourcen aus der Umgebung einbeziehen (surrogate computing, cyber foraging) oder von verfügbaren Cloud-Providern dynamisch Ressourcen allokieren (Mobile Cloud Computing). Eine Übersicht über die möglichen Ressourcen ist in Abbildung 1 dargestellt.

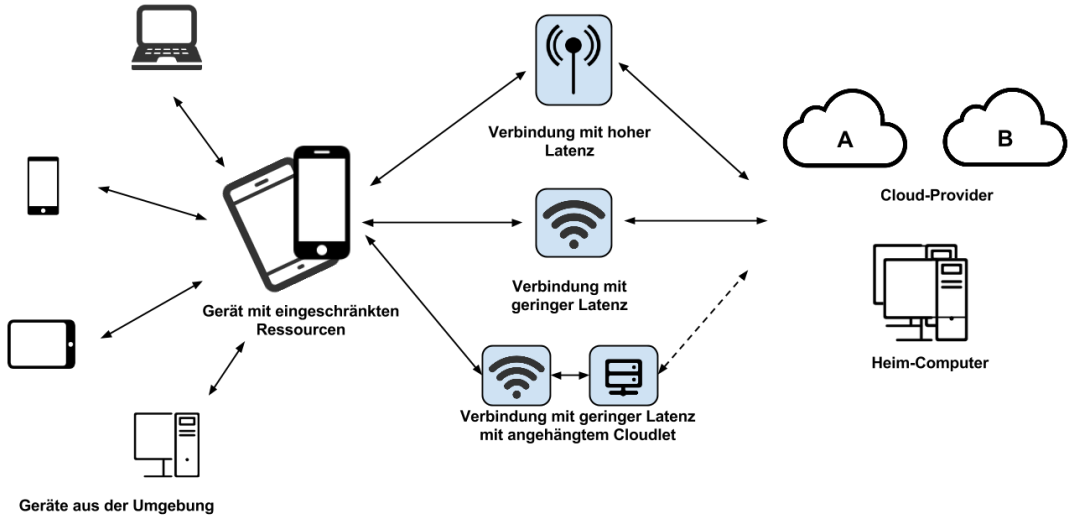

Abbildung 1 - Übersicht über verfügbare Ressourcen

Bestehende Systeme fokussieren sich auf einen bestimmten Ressourcentyp und sind meist beschränkt auf sehr bestimmte Systeme bezüglich:

- Programmiersprache
- Plattform
- Betriebssystem, sogar bestimmte einzelne Versionen von Betriebssystemen

Außerdem wird von keinem der bestehenden Frameworks auf sicherheitsrelevante Punkte eingegangen, was diese für Applikationen, die auf sensiblen Daten operieren bzw. gewisse Sicherheitsanforderungen an die Umgebung stellen, unbrauchbar macht.
Dies ist ein Zustand der in einem so vielseitigen mobilen Markt nicht zielführend ist. Dieses Projekt nimmt sich dieser Problemstellung an und hat das Ziel, eine Architektur und Proof-of-Concept Implementierung eines Cloud-based Mobile Augmentation Frameworks zu erarbeiten. Dabei sollen auf Architekturebene sowie auf Implementierungsebene die Sicherheitsanforderungen erfüllt sein und ein „Lock-in“ auf bestimmte Plattformen bzw. Betriebssysteme vermieden werden. Um dies zu erreichen, wurde eine abstrakte und technologieunabhängige Architektur entwickelt und in weiterer Folge unter Einbeziehung von verfügbaren Technologien verfeinert.
Abgeschlossen wurde das Projekt mit einer wissenschaftlichen Publikation auf die hier auch verwiesen werden kann [1].

2. Architektur

Im Laufe des Projekts wurde eine allgemeine Architektur wie in Abbildung 2 definiert.

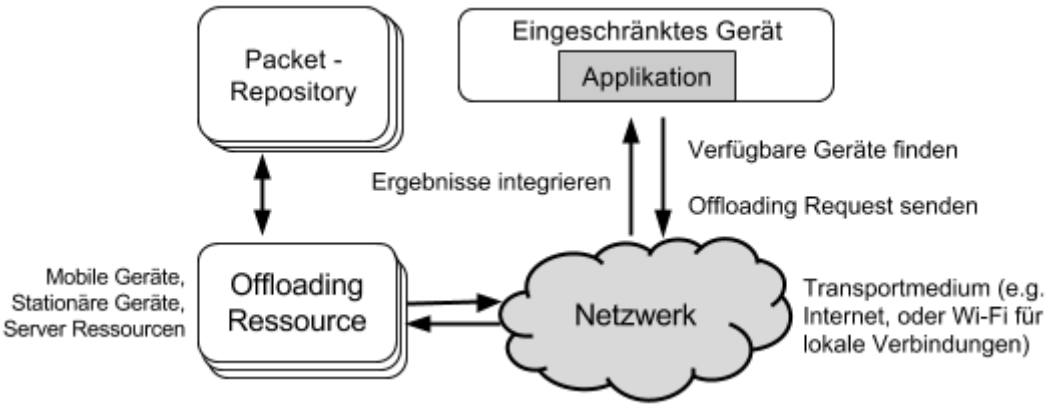

Abbildung 2 - High-level Architektur

Das Packet-Repository beinhaltet Applikationen, inklusive all ihrer notwendigen Abhängigkeiten und diese können von dort von den Ressourcen geladen werden. Repositories werden von mehreren Ressourcen verwendet bzw. können auch auf einzelne Repositories begrenzt werden. Jede Applikation im Repository kann über eindeutige IDs identifiziert werden.
Während der Laufzeit verwaltet das eingeschränkte Gerät eine Liste von verfügbaren Offloading Ressources. Wird vom Framework beschlossen, dass es effizienter ist, eine Operation remote auszuführen, so wird eine Verbindung zu einer verfügbaren Ressource hergestellt und der Offloading-Vorgang initiiert. Die Offloading-Ressource lädt sich im nächsten Schritt die Applikation aus dem Repository, stellt den aktuellen Stand der Applikation wieder her, und führt die gewünschte Operation aus.
Die Architektur ermöglicht prinzipiell zwei Herangehensweisen:

- Der Entwickler bietet Implementierungen für verschiedene Programmierumgebungen an, oder
- der Entwickler bietet eine Implementierung unter Zuhilfenahme von Cross-Platform Umgebungen bzw. allgemein unterstützter Programmiersprachen an.

2.1. Technologiebeobachtungen

Um eine geeignete Technologie zur Umsetzung des beschriebenen Verfahrens auszuwählen, ist es notwendig, den derzeit sehr heterogenen mobilen Markt miteinzubeziehen. Aus Effizienz-gründen ist eine Lösung in nur einer Programmiersprache aber mit Support auf allen gängigen Plattformen wünschenswert. Aktuell treffen diese Anforderungen nur auf JavaScript zu. JavaScript wird prinzipiell auf allen gängigen Plattformen unterstützt (wenn auch mit Unterschieden im Detail der einzelnen Engines), bildet aber dennoch eine gemeinsame Basis. Basierend auf JavaScript wurden mehrere „höhere Sprachen“ entwickelt wie: Coffeescript[2], TypeScript[3] und Dart[4]. Diese Sprachen lassen sich zu effizientem JavaScript Code kompilieren und merzen dabei auch noch die Inkompatibilitäten von unterschiedlichen Javascript Engines aus. Dart geht hierbei sogar noch einen Schritt weiter und stellt ein komplett eigenes Ecosystem, bestehend aus Dart zu JavaScript Compiler, Server Environment und eigener Packet-Verwaltung zur Verfügung. In Kombination mit neuen HTML5 Webtechnologien ist es damit möglich, Cross-Platform Anwendungen (entweder Web-basiert oder unter Verwendung von Cross-Platform Frameworks) zu entwickeln, die nahezu nicht zu unterscheiden sind von nativen Anwendungen.
Basierend auf diesen Technologien wurde eine verfeinerte Version der Architektur entwickelt, wie in Abbildung 3 dargestellt.
Für die Kommunikation ist vorgesehen dass Kommunikationswege mit niedrigen Verzögerungen (WLAN) denen mit potentiell höheren Verzögerungen (3G/4G) vorgezogen werden. Die Architektur zielt sowohl auf entfernte Cloud-basierte Ressourcen, aber auch auf in der Nähe befindliche Geräte ab. Basierend auf unseren Technologieentscheidungen können diese weiter in zwei Gruppen unterschieden werden:

- *Ressourcen mit nativem Dart Support:* Diese Ressourcen erreichen höhere Performance und Effizienz, können aber nur auf eigens dafür ausgelegten Servern eingesetzt werden.
- *Ressourcen mit JavaScript Support:* Diese Ressourcen erreichen nicht die hohen Performancewerte von nativen Ressourcen, es ist aber keine Konfiguration notwendig und

zielt somit auf Ressourcen die von Endbenutzern (z.B. andere mobile Geräte) bereitgestellt werden.

Auch wenn für die Architektur technologiespezifische Annahmen getroffen wurden, kann diese trotzdem auf andere Technologien übersetzt werden. Ein idealer Startpunkt hierfür ist die allgemeine Architektur von Kapitel 2.

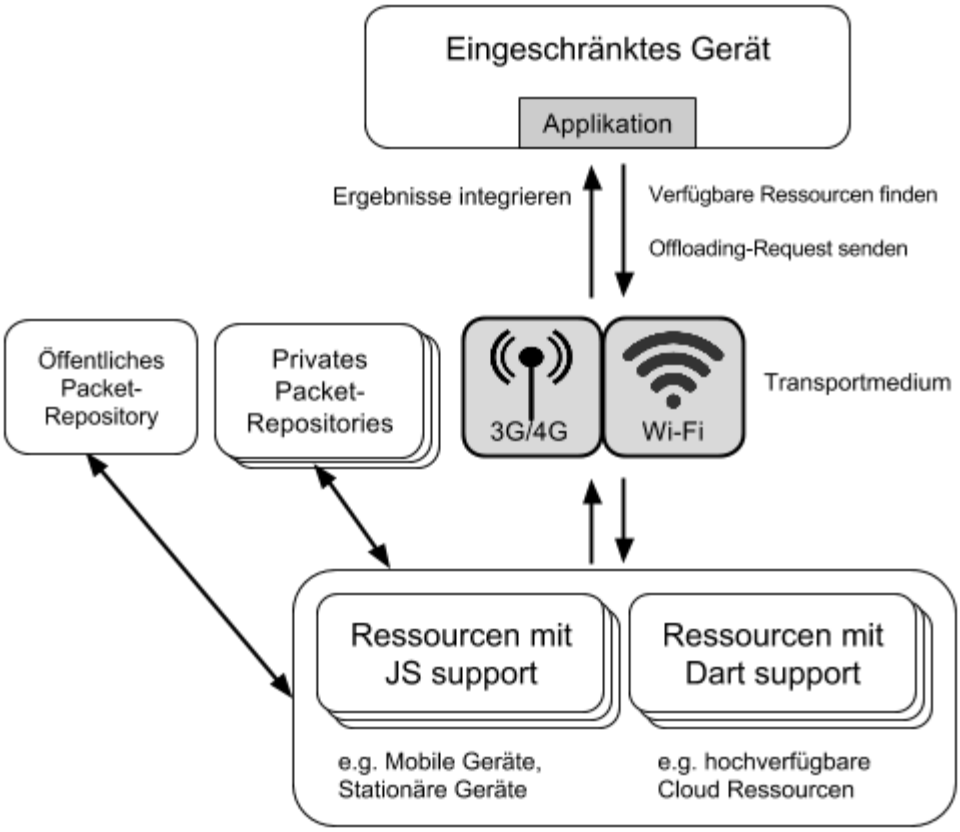

Abbildung 3 - Technologie-spezifische Architektur

2.2. Anforderungsevaluierung

Acht spezialisierte Anforderungen für Cloud-based Mobile Augmentation Systeme wurden bereits in vorhergehenden Projekten definiert: Zuverlässigkeit, Integrität, Datenschutz und Vertraulichkeit, Nachweisbarkeit, Hohes Isolationslevel, Verhindern des Missbrauches von verfügbaren Ressourcen, Interoperabilität für den End-Benutzer, und Interoperabilität für den Ressourcenbetreiber.

Die Evaluierung der Anforderungen gegen die Architektur ergibt folgendes Ergebnis:

- *Zuverlässigkeit*: Ressourcen sollten in Mesh-artigen Strukturen organisiert sein um Single-Points-of-Failure zu vermeiden. Zusätzlich sollte das Package-Repository als zentrale Komponente des Systems mehrfach repliziert werden. Mit diesem Ansatz kann eine hohe Zuverlässigkeit erreicht werden.
- *Integrität*: Aus Sicht des Ressourcenproviders können digitale Signaturen an die Applikationspakete angehängt werden, um deren Echtheit sicherzustellen. Aus Endbenutzer-Sicht kann die Integrität nur über ein gewisses Vertrauen vom Benutzer zum Ressourcenprovider sichergestellt werden. Werden allerdings Ende-zu-Ende Sicherheitsverfahren eingesetzt, so können auch nicht vertrauenswürdige Ressourcen, beispielsweise als Gateway zu vertrauenswürdigen Ressourcen, eingesetzt werden.
- *Datenschutz und Vertraulichkeit:* Es ist Aufgabe der Offloading Engine, die Aufteilung von einzelnen Tasks auf Ressourcen durchzuführen. Diese Aufteilung kann nur korrekt erfolgen (unter der Annahme dass ein Entwickler unterstütztes Verfahren verwendet wird), wenn vom Entwickler die Programmteile korrekt gekennzeichnet werden.
- *Nachweisbarkeit:* Durch das Anbringen von digitalen Signaturen an Offloading-Requests können Benutzer von Ressourcenanbietern eindeutig identifiziert werden. Eine andere und

möglicherweise praktikablere Lösung wäre die Verwendung dezentralisierten Authentifizierungslösungen wie OpenID oder OpenID Connect. Durch beide Verfahren ist diese Anforderung erfüllt.

- *Hohes Isolationslevel:* Im Browser werden bereits fortschrittliche Sandboxing- und Isolationsmethoden eingesetzt, diese bieten daher von Haus aus ein hohes Isolationslevel. In nativen Umgebungen sind noch Verbesserungen notwendig, da Dart keine Sandboxing Mechanismen mitbringt. Aus derzeitiger Sicht ist diese Anforderung also teilweise erfüllt, kann aber durchaus auch durch externe Sandboxing Mechanismen erfüllt werden.
- *Verhindern des Missbrauchs von verfügbaren Ressourcen:* Der Missbrauch von zur Verfügung gestellten Ressourcen kann auf zwei Arten verhindert werden: Entweder durch Aussperren der entsprechenden Benutzer, oder durch Analyse des bereitgestellten Source-Codes, um entsprechende Applikationen zu finden. Ein anderer Weg, um dem Missbrauch entgegen zu treten ist, indem ein Pay-Per-Use Model etabliert wird, in dem ohnehin für alle konsumierten Ressourcen aufgekommen werden muss. Mit einer Anpassung der Strategie kann diese Anforderung also erfüllt werden.
- *Interoperabilität für den End-Benutzer*: Die Technologiespezifische-Architektur basiert auf JavaScript, Dart und HTML5 und ist deshalb auf allen gängigen Plattformen verfügbar.
- *Interoperabilität für den Ressourcen-Betreiber*: Durch die Dart-JavaScript Interoperabilität, kann jeder Benutzer alle Typen an Ressourcen verwenden, außerdem kommen als Ressourcen Cloud-basierte Ressourcen oder vom Benutzer selbst betriebene Server (im native Dart Modus) in Frage, aber auch andere Endgeräte im JavaScript-Modus.

3. Implementierung

Die Implementierung und Funktion der beschriebenen Architektur basiert auf drei grundlegenden Building Blocks: Annotator, Offloading Transformer, Offloading Engine. Durch die Verwendung von Dart kann die Implementierung durch Kompilierung zu JavaScript unverändert auf allen gängigen mobilen Plattformen verwendet und im Browser ausgeführt werden. Nativ kann die Implementierung in Serverumgebungen ausgeführt werden.

3.1. Annotator

Der Annotator bietet ein Attribut, um relevante Teile eines Programms zu kennzeichnen, die in Frage kommen um remote ausgeführt zu werden, wie in Abbildung 4 dargestellt.

```
@OffloadMe(SENSITIVITYLEVEL) Future<String> sample(){
  <your calculation>
}
```

Abbildung 4 – Annotator

Wird dieser Annotator konsequent im entsprechenden Programm angewandt, so wird in weiterer Folge der Offloading Engine eine initiale Aufteilung der Applikation – in Teile die lokal ausgeführt werden müssen und in Teile die remote ausgeführt werden können – geliefert. Zusätzlich kann angegeben werden, ob es sich um eine Berechnung auf sensiblen Daten handelt und somit eine entsprechende Vertrauenswürdigkeit der Ressource benötigt wird. Die endgültige Entscheidung ob die Ausführung eines Programmteils ausgelagert wird hängt aber schlussendlich bei der Offloading Engine.

3.2. Offloading Transformer

Transformer sind ein Konstrukt aus der Dart-Umgebung und dienen dazu, gewisse Aufgaben zur Compilezeit voll automatisiert ablaufen zu lassen. Der Offloading Transformer wird zur Compilezeit aufgerufen und vorverarbeitet den Source-Code der Applikation. Dabei wird der Source-Code dahingehend geändert, dass alle Aufrufe die vom Annotator gekennzeichnet sind, extrahiert werden und so verändert werden, dass der Aufruf über die Offloading Engine geleitet wird und diese den weiteren Ablauf entscheiden kann. Ein einfaches Beispiel, das das Resultat nach einer Transformation zeigt, ist in Abbildung 5 illustriert.

```
Future<String> sample(){
 var m=() { <your calculation> };
 String s="""() { <your calculation source> }""";
 return engine.execute(m, s, [], "sample",
  "example/example.dart", "samplePackage -0.0.1",
  "samplePackage");
}
```

Abbildung 5 - Transformierter Source Code

Zur Laufzeit hat die Offloading Engine die Wahl, den entsprechenden Programmteil direkt auszuführen, oder den Workflow zu ändern und die Ausführung an eine Remote Ressource zu delegieren und erst das Ergebnis wieder zu integrieren und zu retournieren.

3.3. Offloading Engine

Die Offloading Engine ist das Herzstück des Frameworks und führt die eigentliche Ausführung der Programmteile durch. Clientseitig ist die Funktion auf die Kommunikation mit Offloading-Ressourcen beschränkt. Auf Seiten der Ressourcen werden entweder JavaScript Code oder Dart Code ausgeführt. Clientseitig verbindet sich die Offloading Engine zu einem entsprechenden Server und führt die Offloading Operation durch.
Um mit den Ressourcen zu kommunizieren, werden WebSockets und WebRTC verwendet. WebSockets bieten einen asynchronen Kommunikationskanal zwischen Clients und HTTP basierten Servern. WebRTC hingegen ist eine Peer-to-Peer ähnliche Technologie, die eine direkte Kommunikation zwischen Nodes ermöglicht. Dabei können Nodes auf Server bereitgestellte Applikationen, aber auch Webbrowser sein. Die WebRTC Technologie ermöglicht also eine direkte Kommunikation zwischen Endbenutzer und Ressource, wobei dabei die Ausprägung der Ressource (Dart, JavaScript) keine Rolle spielt.
Die Identifizierung von einzelnen Ressourcen kann über zertifikatsbasierte Methoden erfolgen, mit ähnlichen Ansätzen wie Certificate-Pinning.

4. Evaluierung

Die Evaluierung des Frameworks wurde basierend auf zwei Testapplikationen durchgeführt. Die erste Testapplikation bewertet allgemeine Konstrukte, ab welcher Intensität der Aufgabe es lohnenswert ist, diese remote auszuführen. Die zweite Applikation ist eine rechenintensive Gesichtserkennungsapplikation. Das Testsystem besteht aus einem Motorola G2 Smartphone mit Android 5 und aus einem Desktop Computer mit Intel i5 Prozessor mit 3.4 GHz mit Ubuntu 14.04.
Die erste Testreihe evaluiert einen Fibonacci Algorithmus, eine Anwendung die verschachtelte Schleifen verwendet und die Berechnung von SHA1 Hashwerten. Der Wert *n* gibt die Länge der Fibonacci Folge, die Anzahl der Verschachtelungen der Schleife und die Anzahl der durchzuführenden SHA1 Operationen an. Die Ergebnisse sind in Abbildung 6 dargestellt, wobei die Spalte *n* denjenigen Wert angibt, ab dem eine Auslagerung der Operation einen Performancegewinn bringt. Die Ergebnisse zeigen, dass bereits relativ geringe *n*-Werte Verbesserungen bringen.

Benchmark	n	Data Tx (bytes)	Data Rx (bytes)
Fibonacci	23	~450	~80
Nestedloop	16	~450	~80
SHA1	41	~450	~100

Abbildung 6 - Ergebnisse der Evaluierung von Allgemeinen Programmkonstrukten

Die Ergebnisse der Evaluierung der Gesichtserkennungsapplikation sind in Abbildung 7 dargestellt. Hierbei handelt es sich um eine rechenintensive Applikation, und dementsprechend hoch ist der Performancegewinn bei Verwendung des vorgeschlagenen Frameworks. Bei der Gesichtserkennungsapplikation war es nur notwendig eine geringe Menge an Context-Daten zu übertragen, da das entsprechende Bild als Link übertragen wurde. Somit ist eine mehr als 10-fache Beschleunigung der Applikation möglich.

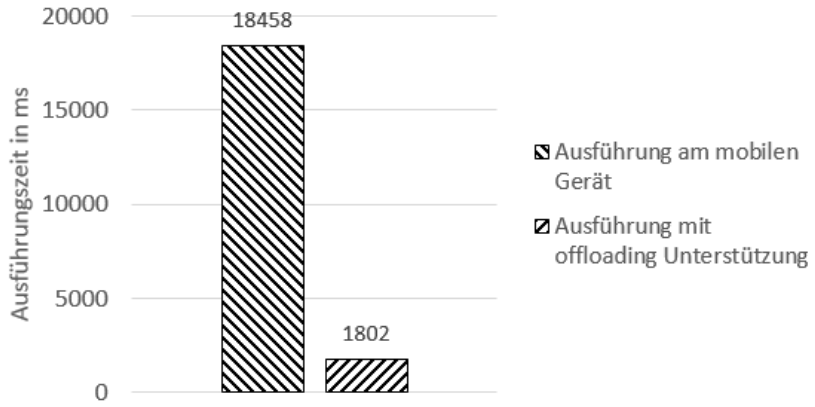

Abbildung 7 - Ergebnisse der Evaluierung einer Gesichtserkennungsapplikation

5. Ergebnisse

In der vorliegenden Arbeit haben wir gezeigt, dass vorhandene Mobile Cloud Computing, Surrogate Computing, Cyber foraging und Cloud-based Mobile Augmentation Frameworks sehr limitiert in ihren Anwendungsgebieten sind, und es dringenden Nachholbedarf in Sicherheitsfragen gibt. Es wurde deshalb eine neue Architektur für ein CMA System vorgeschlagen, die sich diesen Punkten bereits auf der Architekturebene annimmt, was durch die Evaluierung der Anforderungen bestätigt wird.
Die entwickelte Proof-of-Concept Implementierung setzt all diese Punkte um und fokussiert zusätzlich auf die Interoperabilität über Geräte- und Plattformgrenzen hinweg. Die Evaluierungen zeigen die Effizienz des Systems, bereits bei einfachen Aufgaben, aber auch für komplexe Applikationen.

6. Referenzen

[1] A. Reiter and T. Zefferer, “POWER : A Cloud-Based Mobile Augmentation Approach for Web- and Cross-Platform Applications,” in *IEEE CloudNet*, 2015.

[2] “CoffeeScript http://coffeescript.org/, accessed on February 18th 2015.” .

[3] “TypeScript http://www.typescriptlang.org/, accessed on February 18th 2015.” .

[4] “Dart https://www.dartlang.org/, accessed on February 18th 2015.” .