UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

MESTRADO E DOUTORADO EM CIÊNCIAS DA COMPUTAÇÃO

FRANCISCO ANDERSON DE ALMADA GOMES

UM SERVIÇO DE *OFFLOADING* DE DADOS CONTEXTUAIS COM SUPORTE À PRIVACIDADE

FORTALEZA - CEARÁ

2017

FRANCISCO ANDERSON DE ALMADA GOMES

UM SERVIÇO DE *OFFLOADING* DE DADOS CONTEXTUAIS COM SUPORTE À PRIVACIDADE

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Orientador: Prof. Fernando Antonio Mota Trinta, DSc.

Coorientador: Prof. Lincoln Souza Rocha, DSc.

FORTALEZA - CEARÁ
2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

G614s Gomes, Francisco Anderson de Almada.
Um Serviço de Offloading de Dados Contextuais com Suporte à Privacidade / Francisco Anderson de Almada Gomes. – 2017.
95 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2017.
Orientação: Prof. Dr. Fernando Antonio Mota Trinta.
Coorientação: Prof. Dr. Lincoln Souza Rocha.

1. Mobile Cloud Computing. 2. Context-Aware. 3. Middleware. 4. Mobile Device. 5. Offloading. I. Título.

CDD 005

FRANCISCO ANDERSON DE ALMADA GOMES

UM SERVIÇO DE *OFFLOADING* DE DADOS CONTEXTUAIS COM SUPORTE À PRIVACIDADE

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Fernando Antonio Mota Trinta, DSc. (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Windson Viana de Carvalho, DSc.
Universidade Federal do Ceará (UFC)

Prof. Paulo Antonio Leal Rego, DSc.
Universidade Federal do Ceará (UFC)

Prof. Renato de Freitas Bulcão Neto, DSc.
Universidade Federal de Goiás (UFG)

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Primeiramente a Deus por todas as bençãos ao longo da minha vida.

Aos meus pais, Alexandre e Jacqueline, pelo amor, carinho e apoio incondicional.

Ao meu irmão, Alex, pelo apoio e incentivo.

Aos meus orientadores, professor Fernando Trinta e professor Lincoln Rocha, por todo o acompanhamento deste trabalho.

Aos professores Windson, Paulo e Renato, que compõem a banca examinadora e certamente contribuirão para o sucesso da solução desenvolvida.

Ao professor Alexandre Moraes (in memoriam), por toda dedicação com os alunos do Departamento de Engenharia de Teleinformática (DETI).

À Universidade Federal do Ceará, pela oportunidade de fazer o mestrado.

Ao Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat), pelas experiências adquiridas.

À todos os amigos e colegas que acompanharam e compartilharam experiências durante o mestrado. Em especial ao Paulo Artur, Felipe Mota, Alessandro Menezes, Gisele Pereira e Franzé Junior.

À todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“A persistência é o caminho do êxito.”

(Charles Chaplin)

RESUMO

Dispositivos móveis tornaram-se uma ferramenta comum no dia a dia das pessoas. Aplicações móveis cada vez mais exigem o acesso às informações contextuais. Por exemplo, aplicações requerem os dados do ambiente do usuário, bem como dos seus perfis, a fim de se adaptarem (interfaces, serviços, conteúdo) de acordo com esses dados de contexto. Aplicações com esse comportamento são conhecidas como aplicações sensíveis ao contexto. Várias infraestruturas de software foram criadas para ajudar no desenvolvimento dessas aplicações. No entanto, foi verificado que a maioria delas não possui um histórico dos dados contextuais, uma vez que os dispositivos móveis são limitados em recursos de armazenamento. Também foi verificado que a maioria delas não é construída levando em conta a privacidade dos dados contextuais, o que pode levar à exposição desses dados sem o consentimento do usuário. Esta dissertação aborda tais tópicos, estendendo uma plataforma de *middleware* existente que ajuda o desenvolvimento de aplicativos móveis e sensíveis ao contexto. Este trabalho apresenta um serviço denominado COP (***C****ontextual data* ***O****ffloading service with* ***P****rivacy Support*) e é baseado em: (*i*) um modelo de contexto, (*ii*) uma política de privacidade e (*iii*) em políticas de sincronização de dados. O COP visa armazenar e processar os dados contextuais gerados a partir de vários dispositivos móveis, utilizando o poder computacional da nuvem. Para avaliar este trabalho foi desenvolvida uma aplicação que utiliza tanto a migração como o mecanismo de privacidade dos dados contextuais do COP. Outros dois experimentos foram feitos. O primeiro experimento avaliou o impacto da execução de filtros contextuais no dispositivo móvel e no ambiente remoto, em que foi medido o tempo e gasto energético desse processamento. Nesse experimento foi possível concluir que a migração de dados de um dispositivo móvel para um ambiente remoto é vantajosa. O segundo experimento avaliou o gasto energético para o envio dos dados contextuais.

Palavras-chave: Mobile Cloud Computing. Sensibilidade ao Contexto. Middleware. Dispositivo Móvel. Offloading. Privacidade.

ABSTRACT

Mobile devices became a common tool in our daily routine. Mobile applications are demanding access to contextual information increasingly. For instance, applications require user's environment data as well as their profiles in order to adapt themselves (interfaces, services, content) according to this context data. Mobile applications with this behavior are known as context-aware applications. Several software infrastructures have been created to help the development of this applications. However, it was verified that most of them do not store history of the contextual data, since mobile devices are resource constrained. They are not built taking into account the privacy of contextual data either, due the fact that applications may expose contextual data without user consent. This dissertation addresses these topics by extending an existing middleware platform that help the development of mobile context-aware applications. This work present a service named COP (**C**ontextual data **O**ffloading service with **P**rivacy Support) and is based in: (*i*) a context model, (*ii*) a privacy policy and (*iii*) synchronization policies. The COP aims to store and process the contextual data generated from several mobile devices, using the computational power of the cloud. To evaluate this work we developed an application that uses both the migration and the privacy mechanism of the contextual data of the COP. Other two experiments were made. The first experiment evaluated the impact of contextual filter processing in mobile device and remote environment, in which the processing time and energy consumption were measured. In this experiment was possible to conclude that the migration of data from mobile device to a remote environment is advantageous. The second experiment evaluated the energy consumption to send contextual data.

Keywords: Mobile Cloud Computing. Context-Aware. Middleware. Mobile Device. Offloading. Privacy.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AER	Ambiente de Execução Remota
API	*Application Programming Interface*
APK	*Android Application Package*
CAC	Componente de Aquisição Contextual
CAM	*Context Acquisition Manager*
COP	*Contextual data Offloading service with Privacy support*
GPS	*Global Positioning System*
GREat	Grupo de Redes de Computadores, Engenharia de Software e Sistemas
JSON	*JavaScript Object Notation*
LoCCAM	*Loosely Coupled Context Acquisition Middleware*
MAC	*Media Access Control*
MCC	*Mobile Cloud Computing*
MIB	*Management Information Base*
MpOS	*Multiplatform Offloading System*
REE	*Remote Execution Environment*
RPC	*Remote Procedure Call*
RTT	*Round-Trip Time*
SysSU	*System Support for Ubiquity*
UML	*Unified Modelling Language*
WLAN	*Wireless Local Network*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Esta dissertação apresenta uma solução para auxiliar o desenvolvimento de aplicações móveis e sensíveis ao contexto. Em alguns cenários (e.g., monitorar o congestionamento de veículos na cidade), tais aplicações utilizam informações provenientes de diferentes dispositivos para agregá-las com a finalidade de medir, inferir e compreender seu ambiente de execução. Este trabalho propõe um serviço denominado COP (***C****ontextual data* ***O****ffloading service with* ***P****rivacy Support*), para a disseminação de dados contextuais, que baseia-se em: *(i)* um modelo de contexto; *(ii)* uma política de privacidade; e *(iii)* políticas de sincronização.

1.1 Contextualização

Os últimos anos apresentaram uma grande popularização no uso de dispositivos móveis (e.g., *smartphones*, *tablets* e *smartwatches*). Esses dispositivos são equipados com uma quantidade cada vez maior de sensores embarcados (e.g., acelerômetro, giroscópio e *Global Positioning System* (GPS)), os quais possibilitam o sensoriamento de dados do ambiente para que possam ser interpretados e processados por diversas aplicações. Esses dados são chamados de dados contextuais pois são capazes de caracterizar o ambiente em que executam e os usuários que acessam tais aplicações. Segundo Dey, contexto é qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade (DEY, 2001).

A partir da caracterização desse contexto de uso, as aplicações estão aptas a se adaptarem a possíveis mudanças (e.g., alteração da localização do usuário) de forma a oferecerem serviços mais relevantes nessas novas situações contextuais. Esse comportamento adaptativo é chamado de sensibilidade ao contexto e as aplicações dotadas desse comportamento são conhecidas como aplicações móveis e sensíveis ao contexto (PREUVENEERS; BERBERS, 2007). A sensibilidade ao contexto é uma característica chave no cenário de Computação Ubíqua antevisto por Mark Weiser (WEISER, 1999), em que os dispositivos computacionais seriam incorporados em objetos comuns utilizados pelos seres humanos, de uma forma que a interação com o usuário se tornaria natural e quase imperceptível. Muitas das características apresentadas por Weiser são perceptíveis nos *smartphones*, por exemplo, a conectividade.

Por meio da utilização de contexto, a aplicação móvel e sensível ao contexto pode

então executar serviços mais próximos às necessidades do usuário (VIEIRA *et al.*, 2009). Dentre estes serviços, pode-se destacar: (*i*) a assistência na execução da tarefa sendo realizada (e.g., alertar o usuário sobre ações que ele deve executar para alcançar seus objetivos); e (*ii)* adaptação, ou variação do comportamento do sistema, respondendo de forma oportuna às mudanças ocorridas no ambiente e às ações e definições dos usuários (e.g., personalização de interfaces).

Apesar dos benefícios ao utilizar contexto, as aplicações móveis e sensíveis ao contexto, independente do domínio, compartilham da necessidade de realizar uma série de atividades comuns, como por exemplo: a aquisição de dados de múltiplas fontes de contexto, a interpretação de contexto, manipulação de contexto com diferentes níveis de abstração, o controle do comportamento adaptativo da aplicação na presença de uma nova situação contextual, entre outras (SALVIATO, 2012). Levando em conta essas necessidades, ao longo dos anos, várias infraestruturas de suporte (e.g., sistemas de *middleware*, *frameworks* e *toolkits*) ao desenvolvimento e execução de aplicações sensíveis ao contexto foram propostas (YURUR *et al.*, 2014), de forma a simplificar o processo de criação dessas aplicações.

As infraestruturas de suporte proveem funcionalidades que facilitam a aquisição e processamento de dados contextuais e mecanismos que as permitem adaptar-se em função das mudanças contextuais. Por exemplo, uma aplicação de gerenciamento de corrida que faz o registro automático da frequência cardíaca, pressão arterial, velocidade empregada e distância percorrida pelo usuário, poderia utilizar dados de contexto como a temperatura ambiente e a umidade relativa do ar para recomendar aos corredores (i.e., usuários da aplicação), a ingestão regular de líquido (e.g., água ou isotônico) com o intuito de evitar uma possível desidratação. Nesse caso, a aplicação poderia utilizar uma infraestrutura de suporte para monitorar a mudança dessas "variáveis de contexto" e notificá-la quando limiares pré-estabelecidos fossem atingidos, desencadeando assim as recomendações.

1.2 Motivação

As aplicações móveis modernas apresentam uma tendência na troca de informações (e.g., mensagens) em um modelo de rede social (NAVARRO *et al.*, 2015). No modelo de rede social, um enorme conjunto de dados heterogêneos (e.g., multimídia) é gerado. Esses dados podem ser levados em consideração em uma tomada de decisão para aplicações móveis

e sensíveis ao contexto. Além disso, existem aplicações que utilizam dados de mais de um usuário para melhorar a inferência sobre uma nova situação contextual. Assim, essas aplicações necessitam de um grande conjunto de dados para prover serviços diferenciados aos seus usuários.

Levando em consideração o aumento da quantidade dos dados contextuais monitoradas pelas aplicações móveis modernas, o processamento das inferências e o gerenciamento de contexto estão cada vez mais complexos. Apesar dos avanços tecnológicos dos dispositivos móveis tanto nos sistemas operacionais quanto no seu *hardware*, que propiciaram o aumento de desempenho das aplicações móveis, esses dispositivos ainda são limitados se comparados aos *desktops*, já que seu poder computacional é restrito devido a fatores como: tamanho reduzido, menor custo e eficiência energética. Uma possível solução para essa questão é o uso do paradigma denominado de *Mobile Cloud Computing* (MCC) (SHIRAZ *et al.*, 2013; FERNANDO *et al.*, 2013).

De forma sucinta, o paradigma de MCC concentra-se nos benefícios que podem ser alcançados por dispositivos móveis ao se delegar uma operação de armazenamento ou processamento de dados para um ambiente remoto de execução com maior capacidade computacional. Essa técnica de delegação de armazenamento e processamento é referenciada na literatura como *offloading* (SANAEI *et al.*, 2014).

Particularmente, o *offloading* de dados precisa lidar com questões sensíveis relativas à privacidade dos dados dos usuários (e.g., Quais dados devem ser anonimizados? Quem pode ter acesso aos dados? O acesso será dado no todo ou em parte?). Quando negligenciadas, tais questões podem acarretar na divulgação pública dos dados contextuais do usuário sem a sua devida anuência, pavimentando um caminho para onerosas disputas judiciais. Para Nissenbaum (2004), a migração de dados contextuais diminui a confiança no sistema. Por exemplo, dados contextuais de localização do usuário podem ser compartilhados com segurança com colegas de trabalho durante o expediente, porém podem prejudicar a confiança no sistema se esses dados forem compartilhados com as mesmas pessoas durante a noite. Assim, o sentimento de privacidade dos usuários está relacionado com o controle que eles têm sobre seus dados. A privacidade em aplicações móveis e sensíveis ao contexto é uma preocupação crescente tanto para os usuários como para os desenvolvedores (TOCH, 2014). Nos sistemas operacionais móveis atuais, o controle sobre o contexto de localização é extremamente limitado, permitindo basicamente que os usuários autorizem ou não que um aplicativo acesse a *Application Program-*

ming Interface (API) de localização. Nessa abordagem de "tudo ou nada" para a utilização do contexto de localização, os usuários não podem diferenciar entre local privado e local público. Portanto, é necessário que a infraestrutura de gerenciamento de contexto possua um mecanismo de privacidade para que o usuário possa tornar públicos ou não os seus dados contextuais para outros usuários.

A partir do exposto é apresentada a seguinte questão de pesquisa deste trabalho:

- **Questão de Pesquisa**: Como o uso de técnicas de *offloading* podem melhorar ou enriquecer as infraestruturas de suporte sensíveis ao contexto no que diz respeito à migração de dados do dispositivo móvel para uma infraestrutura de nuvem?

A partir da questão de pesquisa foram levantadas duas hipóteses para avaliação:

- **Hipótese 1**: A migração de dados contextuais de vários dispositivos móveis para nuvem pode melhorar a caracterização de uma dada situação contextual. Com a migração destes dados, é possível realizar agregações considerando dados de vários dispositivos;
- **Hipótese 2**: A utilização da técnica de *offloading* pode diminuir o tempo e o consumo energético na recuperação dos dados contextuais para a aplicação móvel.

1.3 Objetivos e Contribuições

Com o intuito de mitigar os problemas apresentados, este trabalho de mestrado tem como objetivo desenvolver um serviço de *offloading* de dados que permita migrar dados contextuais do dispositivo móvel para uma infraestrutura de nuvem e permita que o uso desses dados seja o mais transparente possível para o desenvolvedor, assim como fornecer um suporte à privacidade desses dados. Esse serviço é denominado de *Contextual data Offloading service with Privacy support* (COP), que é uma extensão de um *middleware* sensível ao contexto já existente, denominado de *Loosely Coupled Context Acquisition Middleware* (LoCCAM) (MAIA *et al.*, 2013). Na extensão do LoCCAM, foram incluídas funcionalidades que permitem lidar com:

- O armazenamento dos dados contextuais gerados pelos usuários do sistema;
- Disseminação de dados contextuais entre o dispositivo móvel e o ambiente de

nuvem de forma transparente e configurável; e

- A privacidade dos dados contextuais de forma ajustável, para restringir a divulgação e difusão de dados contextuais de cada usuário do sistema.

O armazenamento dos dados contextuais é necessário tanto para aplicações que utilizam o histórico dos usuários (e.g., a localização de um usuário nos últimos 30 dias) como para aplicações que necessitam do compartilhamento dos dados dos usuários do sistema (e.g., o congestionamento de veículos em uma determinada cidade). Uma das limitações do dispositivo móvel é a capacidade de armazenamento. Portanto, é fundamental que ocorra a disseminação de dados entre o dispositivo móvel e a infraestrutura de nuvem. A privacidade dos dados contextuais é importante para evitar que, durante a disseminação de tais dados, o usuário torne pública suas informações sem a sua devida anuência.

Além do LoCCAM, o COP utiliza um *framework* de suporte ao *offloading* denominado de *Multiplatform Offloading System* (MpOS) (COSTA *et al.*, 2015), para executar o processamento das requisições de recuperação dos dados contextuais presentes na nuvem para o dispositivo móvel. Para validar a solução, o COP foi submetido a um conjunto de testes, em que foi verificado ganhos na capacidade de processar dados com o mínimo de impacto possível no desempenho e consumo energético no dispositivo móvel.

1.4 Metodologia

A metodologia utilizada neste trabalho apresenta as seguintes etapas:

- **Revisão da Literatura**: Foi realizada uma revisão bibliográfica envolvendo os conceitos de Sensibilidade ao Contexto e *Mobile Cloud Computing* (MCC). Também foi realizada uma revisão da literatura de trabalhos que envolviam conjuntamente a sensibilidade ao contexto com nuvem;
- **Estudo dos Trabalhos Relacionados**: O escopo foi reduzido para se focar na busca por trabalhos relacionados a soluções que auxiliam o desenvolvimento de aplicações móveis e sensíveis ao contexto com migração de dados. Os trabalhos encontrados foram estudados;
- **Definição das Funcionalidades**: Foi definido um conjunto de funcionalidades necessárias ao serviço proposto com base nas características encontradas nos

trabalhos relacionados;

- **Modelagem**: Nesta etapa foi definida a representação de contexto utilizada no COP, baseada no modelo de contexto definido para o LoCCAM. Também foi definida nessa etapa, a política de privacidade para a solução proposta com o objetivo de evitar a disseminação dos dados privados dos usuários. Por fim, foi definido em que momentos os dados contextuais devem ser migrados do dispositivo para nuvem. Para a migração desses dados, foram estabelecidas diferentes estratégias de envio, chamadas de políticas de sincronização;
- **Implementação**: Uma vez que foram definidas suas funcionalidades, o modelo de contexto, a política de privacidade e as políticas de sincronização, pode-se partir para a implementação do serviço proposto;
- **Avaliação**: Com o serviço implementado, pode-se realizar uma prova de conceito para verificar se o COP atende aos objetivos propostos inicialmente. A solução também foi avaliada em termos de desempenho e consumo energético com objetivo de mostrar o impacto da migração de dados contextuais. Os resultados encontrados durante a avaliação mostram que é vantajoso realizar essa migração de dados tanto em termos de diminuir o espaço de armazenamento necessário no dispositivo móvel, como também em se obter ganhos de desempenho e menor gasto energético na recuperação desses dados.

1.5 Organização da Dissertação

Esta dissertação está organizada em seis capítulos. Este capítulo descreveu uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, os objetivos, as contribuições e a metodologia deste trabalho.

O Capítulo 2 trata da fundamentação teórica, apresentando os conceitos importantes adotados na pesquisa. Dentre eles, pode-se citar: a Sensibilidade ao Contexto e o paradigma de MCC.

O Capítulo 3 apresenta os trabalhos relacionados ao serviço proposto. Foram analisados trabalhos que propõem infraestruturas de suporte sensíveis ao contexto, que permitem o compartilhamento de dados contextuais, e apresentem pelo menos um dos dois critérios a seguir: suporte à configuração da migração dos dados contextuais ou capacidade de fornecer algum tipo

de privacidade para esses dados.

No Capítulo 4 o serviço proposto é descrito, bem como os modelos de contexto e privacidade adotados, além das políticas de sincronização utilizadas. Também é apresentado um cenário motivador para o trabalho.

Por meio da implementação de uma prova de conceito, o Capítulo 5 apresenta exemplos de código de utilização do serviço proposto. Neste capítulo também são detalhados as metodologias, procedimentos e resultados obtidos tanto na avaliação do mecanismo de recuperação dos dados contextuais como na avaliação do envio desses dados do dispositivo móvel para a nuvem. Também é apresentado um comparativo entre a solução proposta e os trabalhos relacionados.

Por fim, o Capítulo 6 descreve os resultados alcançados por este trabalho, assim como as suas conclusões e as publicações resultantes. Além disso, são apresentadas as limitações do COP e as possíveis melhorias a serem consideradas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos e definições relacionados às subáreas que fundamentam este trabalho. A Seção 2.1 apresenta conceitos relacionados a Sensibilidade ao Contexto e discute sobre o suporte ao desenvolvimento de aplicações sensíveis ao contexto. A Seção 2.2 apresenta o *middleware* LoCCAM, expondo a arquitetura, componentes e estratégias utilizadas. A Seção 2.3 apresenta os conceitos do paradigma de *Mobile Cloud Computing*, explicando seus princípios, as arquiteturas e os desafios da área. Por fim, a Seção 2.4 apresenta o *framework* MpOS, que realiza a operação de *offloading* em múltiplas plataformas móveis, dando uma visão geral dessa solução.

2.1 Sensibilidade ao Contexto

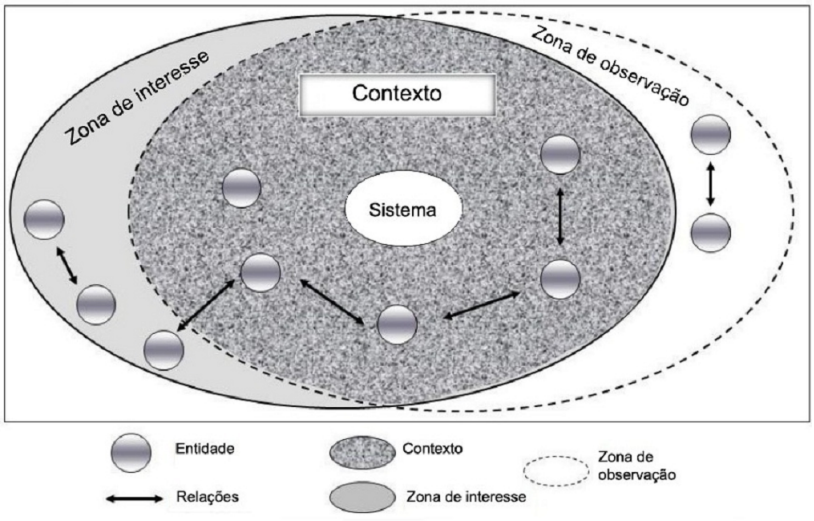
Sensibilidade ao contexto é uma tema estudado há décadas, sendo apresentado a primeira vez por Schilit e Theimer (SCHILIT; THEIMER, 1994). Para esses autores, a sensibilidade ao contexto é definida como "a habilidade de uma aplicação móvel em descobrir e reagir a mudanças no ambiente em que ele está situado".

A definição de contexto mais utilizada é a de Dey (DEY, 2001), cujo foco está na interação do usuário com seu aplicativo, seu ambiente e a situação dos recursos envolvidos. De acordo com o autor, contexto é "qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerada relevante para a interação entre um usuário e uma aplicação, incluindo o usuário e a aplicação em si".

O conceito de Dey foi expandido por Viana (VIANA, 2010) de maneira a retirar a restrição da definição que limitava contexto aos elementos que caracterizavam a interação entre usuário e sistema. Segundo a definição de Viana (VIANA, 2010), os elementos que compõem o contexto (e.g., temperatura ambiente e localização do usuário) são definidos baseados na relevância que possuem para o sistema e na possibilidade do sistema sensoreá-los em um determinado instante do tempo. A ideia visa incluir sistemas que fazem a aquisição de dados contextuais relevantes no instante do sensoriamento, mas que podem ser utilizados depois para melhoria de funcionalidades do sistema (e.g., um sistema móvel de anotação de fotos).

Nessa definição, contexto é definido como a interseção entre dois conjuntos dinâmicos e evolutivos, como mostra visualmente a Figura 1. O primeiro conjunto é chamado de Zona de Interesse (ZI), e é composto de todos as informações do ambiente, do sistema e do usuário que o sistema gostaria de conhecer. O segundo conjunto é chamado de Zona de Observação (ZO), e é composto de todas as informações do ambiente, do sistema e do usuário que o sistema é capaz de fornecer. A interseção entre o que é de interesse do sistema e o que o sistema consegue obter em um instante *t* é considerado contexto. Assim, a sensibilidade ao contexto é a capacidade das aplicações monitorarem as informações do usuário e do ambiente em que ele está inserido e alterar o seu comportamento em virtude de mudanças nessas informações. Essa definição é adotada no LoCCAM, que é a infraestrutura de suporte ao desenvolvimento de aplicações contextuais estendida pelo COP, e por esse motivo, também adotada na solução proposta.

Figura 1 – Contexto como Interseção da ZI e ZO



Fonte: Adaptado de (VIANA, 2010)

Em um sistema ubíquo, a sensibilidade ao contexto pode ser considerada um dos requisitos básicos (LIMA *et al.*, 2011). Realmente, as aplicações dotadas dessa característica fazem parte do cenário da Computação Ubíqua imaginado por Mark Weiser (WEISER, 1999). Nesse cenário, os dispositivos computacionais seriam incorporados em objetos comuns utilizados pelos seres humanos, de uma forma que a interação com o usuário se tornaria natural e quase imperceptível. Um dos princípios identificados por Weiser para a Computação Ubíqua, que são perceptíveis nos *smartphones* e *tablets* atuais, é a conectividade.

2.1.1 Desenvolvimento de Aplicações Sensíveis ao Contexto

Existem desafios no desenvolvimento de uma aplicação sensível ao contexto (SALVIATO, 2012). Dentre esses desafios, pode-se destacar: (*i*) a caracterização de contexto para a utilização nas aplicações, assim como a representação de um modelo; (*ii*) a aquisição contextual a partir de várias fontes heterogêneas; (*iii*) o processamento e interpretação dos dados contextuais adquiridos; (*iv*) a disseminação e compartilhamento dos dados contextuais entre as aplicações; (*v*) a adaptação da aplicação de acordo com contexto.

As aplicações móveis sensíveis ao contexto possuem desafios que são herdados da natureza do local em que elas são executadas, no caso o dispositivo móvel. Apesar dos avanços tecnológicos desse dispositivo, tanto nos sistemas operacionais como no *hardware*, estes ainda são limitados tanto em processamento computacional quanto em armazenamento de dados. Devido a limitação de armazenamento de dados, aplicações móveis sensíveis ao contexto que utilizam o histórico dos usuários ou necessitam do compartilhamento dos dados dos usuários do sistema podem ser afetadas.

Um desafio que também pode ser encontrado no desenvolvimento de aplicações móveis sensíveis ao contexto é a privacidade dos dados contextuais. Nas plataformas móveis atuais, o controle sobre os dados contextuais são limitados. Em geral, usuários autorizam ou não que as aplicações acessem as APIs dessas plataformas. Por exemplo, se a aplicação precisa acessar a localização do usuário, não é possível diferenciar entre um local privado, que o usuário não deseja enviar a localização, e um local público. A privacidade dos dados contextuais é importante para evitar que durante a disseminação desse dados, o usuário torne pública suas informações sem a sua autorização.

Para facilitar o desenvolvimento, a extensibilidade e reusabilidade em sistemas sensíveis ao contexto, várias infraestruturas de suporte (e.g., sistemas de *middleware*, *frameworks* e *toolkits*) foram propostas (KNAPPMEYER *et al.*, 2013; YURUR *et al.*, 2014; BALDAUF *et al.*, 2007).

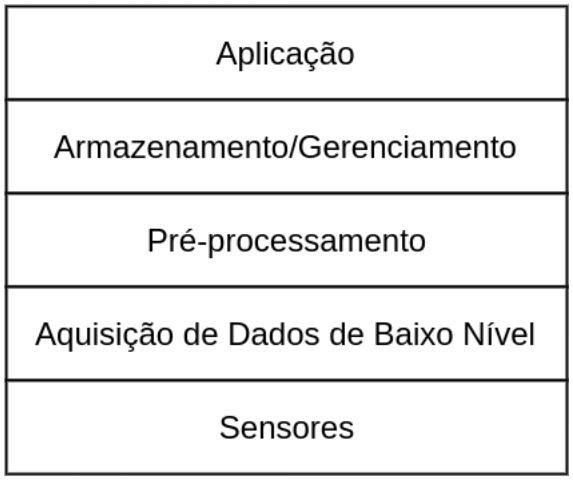
2.1.2 Infraestruturas de Suporte

As infraestruturas de suporte sensíveis ao contexto fornecem funcionalidades básicas relativas ao gerenciamento dos dados contextuais, para que as aplicações possam fazer uso

desses serviços, facilitando o desenvolvimento dos sistemas. Essas infraestruturas fornecem mecanismos para aquisição de dados contextuais do ambiente, geram informações de mais alto nível a partir dos dados contextuais sensoreados por meio de inferências e agregações, realizam o armazenamento do contexto, entre outras funcionalidades. Algumas infraestruturas utilizam uma arquitetura cliente/servidor, em que nenhuma ou poucas tarefas de gerenciamento de contexto são executadas no dispositivo móvel (KNAPPMEYER *et al.*, 2013). Assim as aplicações móveis e sensíveis ao contexto compartilham um conjunto de servidores remotos de gerenciamento de contexto, permitindo que os usuários consumam serviços personalizados e compartilhem suas informações entre eles. As plataformas de *middleware*, que fazem parte dessas infraestruturas, funcionam como uma camada intermediária entre o sistema operacional e a aplicação, visando prover interoperabilidade e diminuir a complexidade no desenvolvimento de aplicações sensíveis ao contexto.

Apesar de peculiaridades presentes em cada infraestrutura de suporte, as funcionalidades básicas relativas ao gerenciamento de contexto podem ser mostradas em uma arquitetura conceitual. A Figura 2 proposta em (BALDAUF *et al.*, 2007) mostra uma arquitetura em camadas de uma infraestrutura de suporte sensível ao contexto.

Figura 2 – Arquitetura de Referência para Infraestruturas de Suporte Sensíveis ao Contexto



Fonte: Adaptado de (BALDAUF *et al.*, 2007)

A arquitetura apresentada em (BALDAUF *et al.*, 2007) é dividida em 5 camadas:

- **Sensores**: A primeira camada consiste em uma coleção de diferentes sensores. A palavra "sensor" não significa apenas sensoriamento de *hardware*, mas qualquer fonte de dados que podem fornecer dados contextuais úteis. Esses sensores

podem ser classificados em: físicos, lógicos ou virtuais (INDULSKA; SUTTON, 2003);

- **Aquisição de Dados de Baixo Nível**: A segunda camada é responsável por recuperar os dados de baixo nível oriundos da primeira camada. Esses dados são obtidos usando *drivers* apropriados para sensores físicos e APIs para sensores lógicos e virtuais;
- **Pré-processamento**: A terceira camada não é implementada em todo sistema sensível ao contexto. Em sistemas que consistem de diferentes fontes de contexto, essa camada permite que diferentes dados contextuais possam ser combinados para um dado de mais alto nível, por meio de inferências e agregações;
- **Armazenamento/Gerenciamento**: A quarta camada é responsável por organizar todos os dados adquiridos das camadas inferiores e fornecer-los através de uma interface amigável às aplicações. Em geral, as aplicações podem obter acesso aos dados de duas maneiras: síncrona ou assíncrona; e
- **Aplicação**: A quinta camada é a responsável por de fato oferecer os serviços relevantes ao usuário de acordo com a reação e adaptação aos dados contextuais sensoreados.

No LoCCAM, todas as camadas dessa arquitetura de referência são implementadas, porém com a ressalva que a camada de pré-processamento é delegada para o desenvolvedor de componentes responsáveis por fornecer os dados contextuais. Assim, essa arquitetura de referência é também adotada no trabalho proposto.

2.1.3 Modelos de Contexto

No desenvolvimento de aplicações móveis e sensíveis ao contexto é importante definir como os dados contextuais serão modelados. Um modelo de contexto é necessário para representar e armazenar os dados contextuais no sistema. Existem várias abordagens de modelagem de contexto, que se baseiam na forma como que os dados são representados no sistema (PERERA *et al.*, 2014). Essas abordagens são descritas a seguir:

- **Chave-Valor**: é a representação mais simples de modelagem de contexto. Nessa modelagem, a consulta aos dados contextuais é realizada através do casamento dos pares chave-valor. Apesar da simplicidade, essa abordagem possui dificul-

dade em expressar detalhes dos dados contextuais, além de ser pouco formal. Esse modelo é bastante utilizado nas infraestruturas de suporte;

- **Linguagem de Marcação**: os modelos baseados em marcação usam uma estrutura de dados hierárquico, que consiste na marcação de *tags* com atributos e conteúdo (e.g., *eXtensible Markup Language* (XML));
- **Gráficos**: essa modelagem consiste em representar os dados contextuais graficamente (e.g., *Unified Modelling Language* (UML));
- **Orientados a Objetos**: essa modelagem de contexto utiliza técnicas de orientação a objetos (e.g., encapsulamento, herança). As abordagens usam vários objetos para representar diferentes tipos de contexto (e.g., temperatura, localização) e encapsulam os detalhes do processamento e representação do contexto.
- **Baseados em Lógica**: essa representação possui um alto grau de formalidade. Tipicamente, fatos, expressões e regras são usadas para definir o modelo de contexto. Essa abordagem apresenta um maior grau de complexidade na representação.
- **Baseados em Ontologias**: ontologias representam uma descrição dos conceitos e relacionamentos. Essa abordagem possui uma alta e formal expressividade dos dados contextuais;

A seguir é apresentado o *middleware* LoCCAM, que é uma infraestrutura de suporte sensível ao contexto desenvolvido no laboratório Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat), que é um dos pontos de partida do COP.

2.2 LoCCAM

O LoCCAM é um *middleware* que dá suporte ao desenvolvimento e execução de sistemas sensíveis ao contexto na plataforma Android (MAIA *et al.*, 2013). Esse *middleware* intermedia de forma adaptativa a aquisição de contexto, utilizando os sensores presentes no próprio dispositivo móvel.

No LoCCAM, cada elemento responsável pela captura dos dados contextuais (i.e., sensores) é batizado de Componente de Aquisição Contextual (CAC). Esses dados são disponibilizados pelo LoCCAM em um espaço de tuplas, no qual as aplicações podem fazer consultas diretas (i.e., interação do tipo *request-response*) ou subscreverem interesse em um determinado

tipo de dado contextual. O segundo caso segue um modelo de interação do tipo *publish-subscribe*, onde as aplicações são notificadas quando os dados de contexto de interesse são inseridos no espaço de tuplas. A camada responsável pela aquisição de contexto e a camada de aplicação são desacopladas. Desse modo, a obtenção dos dados contextuais ocorre de forma transparente, na qual as aplicações não necessitam conhecer a origem dos dados contextuais para utilizá-los. Além disso, com o objetivo de otimizar o uso dos recursos computacionais do dispositivo móvel, o LoCCAM adapta, em tempo de execução, a sua camada de aquisição de contexto, gerenciando o ciclo de vida dos CACs em função dos interesses das aplicações (e dos próprios CACs) em determinados tipos de dados de contexto.

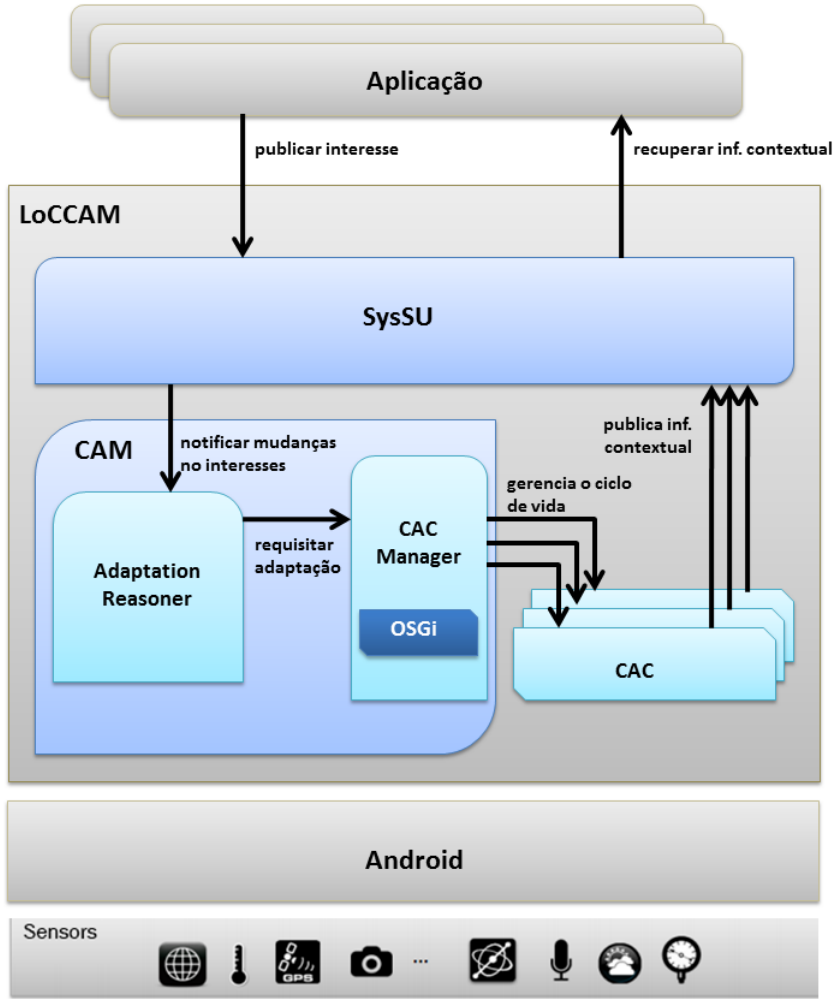
2.2.1 Arquitetura

O LoCCAM pode ser dividido em duas partes principais: o *framework Context Acquisition Manager* (CAM) e o módulo *System Support for Ubiquity* (SysSU) (LIMA *et al.*, 2011). Uma visão geral da arquitetura do LoCCAM é apresentada na Figura 3. O *framework* CAM é dividido em: *Adaptation Reasoner* e CAC *Manager*. O *Adaptation Reasoner* é o responsável por manter uma lista com a relação de interesses de todas as aplicações, por verificar eventuais modificações que possam ocorrer nessa relação de interesses e, caso seja necessário, realizar a adaptação devida para satisfazer os novos interesses ou economizar algum recurso computacional. Uma vez estabelecida a adaptação a ser executada (i.e., qual CAC será ativado ou desativado), o CAC *Manager*, que é responsável pelo gerenciamento do ciclo de vida do CAC, assume a função de executá-la.

O módulo SysSU é uma infraestrutura que tem por objetivo dar suporte a interação espontânea em ambientes ubíquos (LIMA *et al.*, 2011). Entre as funcionalidades do módulo, é permitido que dados contextuais, no formato de tuplas (Definição 1), sejam armazenados e acessados de maneira síncrona e assíncrona. Embora o SysSU tenha sido projetado para ter um funcionamento distribuído, no LoCCAM ele foi adaptado para funcionar embarcado e em um único dispositivo móvel.

Definição 1 (Tupla) *Uma tupla t é uma sequência de n pares, tal que t = ⟨ (n_0, v_0), (n_1, v_1), ..., (n_n-1, v_n-1)⟩, onde cada par* (n_i, v_i) *é formado por um nome* (n_i) *e um valor* (v_i)*. Cada par é também único e* $\forall (n_j, v_j), (n_k, v_k) \in t, j \neq k$ *temos que* $n_j \neq n_k$*.*

Figura 3 – Arquitetura do LoCCAM



Fonte: Adaptado de (DUARTE *et al.*, 2015)

Toda a interação entre as aplicações e o LoCCAM ocorre por meio do SysSU. É nele que todos os CACs publicam os dados contextuais sensoreados, bem como as aplicações e/ou CACs podem realizar consultas síncronas ou assíncronas. Assim, esse módulo é um intermediador entre o LoCCAM, as aplicações e os CACs. Quando as aplicações e/ou os CACs registram no SysSU os interesses em um determinado tipo de dado contextual, o módulo notifica o *Adaptation Reasoner*, que por sua vez, estabelece quais adaptações devem ser feitas na camada de aquisição de contexto que, em seguida, serão executadas pelo CAC *Manager*. Além disso, o SysSU é responsável por possibilitar a interoperabilidade entre os diferentes componentes que compõem o sistema e permitir o desacoplamento entre as camadas do *middleware*.

2.2.2 Componentes de Aquisição Contextual

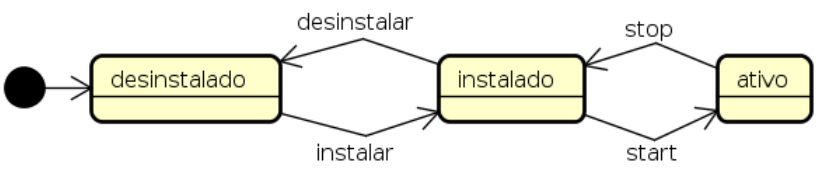
A unidade básica do *middleware* é o Componente de Aquisição de Contexto (CAC). Esse componente adquire todo dado contextual, que então é publicado no espaço de tuplas. Assim, as aplicações não precisam se preocupar com essa aquisição.

Cada CAC é responsável por encapsular um sensor presente no dispositivo móvel, sendo que este pode ser físico, lógico ou virtual. Os sensores físicos são os que adquirem dados diretamente de sensores fornecidos pelo dispositivo móvel (e.g., temperatura e umidade relativa do ar). Os sensores lógicos são os que utilizam dados gerados por outros sensores (físicos ou virtuais) para realizar inferências e gerar um novo dado contextual (e.g., sensor que faz uma relação entre a temperatura e umidade relativa do ar para gerar um dado de risco de desidratação). Os sensores virtuais são os que adquirem dados contextuais a partir de outros sistemas de software ou serviços (e.g., um componente que acessa um serviço *web* meteorológico para fornecer informações climáticas).

Como dito anteriormente, o CAC *Manager* é responsável pelo gerenciamento do ciclo de vida de um CAC. A Figura 4 apresenta esse ciclo de vida, que pode assumir três estados: desinstalado, instalado e ativo.

No momento em que um CAC é adicionado ao repositório local do dispositivo móvel, ele se encontra “desinstalado”, sendo esse o estado padrão. A partir desse ponto, o CAC pode ir para o estado “instalado”. Quando instalado, o CAC *Manager* pode ativar o CAC de acordo com o interesse das aplicações. Quando ele está no estado “ativo”, a aquisição de contexto está em execução e dados contextuais estão sendo publicados no espaço de tuplas. A qualquer momento, se for de interesse da aplicação, o CAC *Manager* pode parar o CAC e o estado do componente volta para “instalado”.

Figura 4 – Ciclo de Vida do CAC



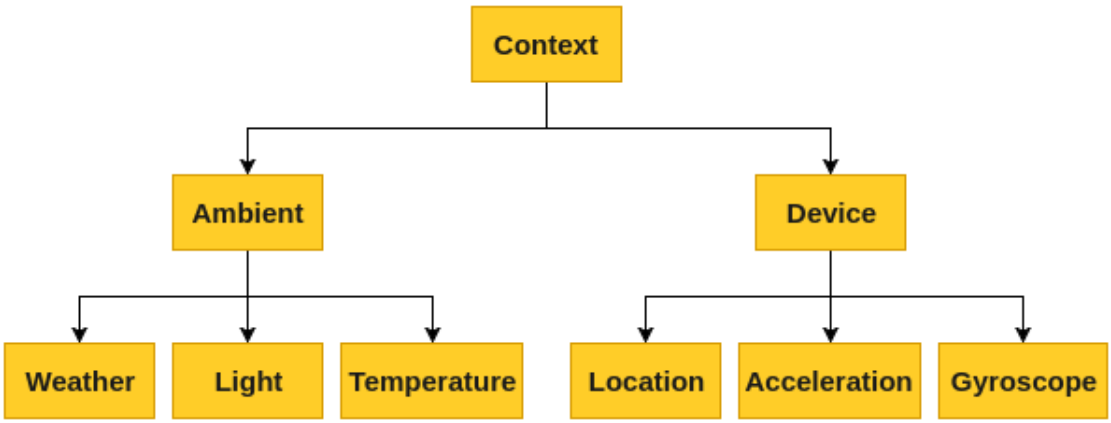
Fonte: Adaptado de (FONTELES, 2013)

2.2.3 Representação do Modelo de Contexto

Para a comunicação entre o *middleware* e os aplicativos é utilizado um vocabulário compartilhado, e que representa os tipos de dados contextuais. Cada tipo de dado contextual está associado univocamente à uma chave, denominada *Context Key* (CK). Internamente ao LoCCAM, essa chave é utilizada para mapear o CAC responsável pelo provimento do dado contextual. Além disso, as CKs podem ser utilizadas pelas aplicações e outros CACs para consulta de dados contextuais ou subscrição em eventos relacionados a esse mesmo tipo de dado contextual. Para a geração de chaves únicas, foi definido um esquema hierárquico inspirado na Base de Gerenciamento de Informação (do inglês, *Management Information Base* (MIB))[1], que identifica o tipo do dado contextual a ser recuperado. Usando esse esquema, um dado é referenciado através de uma sequência de nomes separados por pontos. Por exemplo, a CK "`context.device.location`" é utilizada no LoCCAM para identificar dados de localização do dispositivo.

As CKs seguem a árvore de hierarquia apresentada na Figura 5. Quando novos CACs são desenvolvidos, a hierarquia pode ser estendida a partir de qualquer nó. Por exemplo, supondo que seja necessário um CAC que realize a aquisição de dados contextuais do usuário, mais especificamente a idade dele. Então seria adicionado um novo nó "`user`" abaixo de "`context`" e um nó "`age`" abaixo de "`user`". Assim, a CK ficaria "`context.user.age`".

Figura 5 – Árvore de Hierarquia das CK



Fonte: Elaborado pelo autor

No LoCCAM, as tuplas armazenadas no espaço de tuplas possuem um formato

[1] http://www.ieee802.org/1/pages/MIBS.html

padrão (i.e., o modelo de contexto definido) e são representadas como descrito na Definição 2.

Definição 2 (Tupla no LoCCAM) *Uma tupla no LoCCAM é representada por 5 elementos* $t = \langle(\texttt{contextkey}, \text{“?”}), (\texttt{source}, \text{“?”}), (\texttt{values}, \text{“?”}), (\texttt{accuracy}, \text{“?”}), (\texttt{timestamp}, \text{“?”})\rangle$*, onde* `contextkey` *é utilizado para identificar o tipo de dado contextual;* `source` *é utilizado para informar a fonte do dado (i.e., sensor físico, lógico ou virtual);* `values` *contém o valor do dado contextual;* `accuracy` *informa a precisão de uma leitura; e* `timestamp` *contém o instante de tempo, em milissegundos, em que o dado foi sensoreado.*

A estrutura apresentada em (2.1) caracteriza uma leitura do dado de contexto “temperatura do ambiente” no formato de tupla do LoCCAM. Esse dado foi fornecido por um sensor físico e possui valor de 25 graus Celsius.

$$\begin{aligned} t = \langle &(\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”}), \\ &(\texttt{source}, \text{“}\texttt{physical}\text{”}), \\ &(\texttt{values}, \text{“}25\text{”}), \\ &(\texttt{accuracy}, \text{“}0\text{”}), \\ &(\texttt{timestamp}, \text{“}239459060969\text{”})\rangle \end{aligned} \tag{2.1}$$

2.2.4 Filtros Contextuais

O LoCCAM recupera os dados contextuais no espaço de tuplas do SysSU por meio de filtros contextuais. Duas formas de recuperação estão presentes: (i) *pattern matching*; e (ii) *pattern matching* com filtro de tuplas. Na primeira, um subconjunto dos campos (nome e valor) de uma tupla do LoCCAM (Definição 2) é fornecido e todas as tuplas existentes no espaço de tuplas que casam com o padrão fornecido são retornadas. Por exemplo, caso uma aplicação queira acessar a temperatura do ambiente onde o dispositivo móvel se encontra, a aplicação pode fazer o uso de um padrão de tuplas, como descrito em (2.2). Caso a mesma aplicação tenha interesse em ler tuplas que representem a localização do dispositivo móvel, o padrão (2.3) deve ser utilizado.

$$p = \langle(\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”})\rangle \tag{2.2}$$

$$q = \langle(\texttt{contextkey}, \text{“}\texttt{context.device.location}\text{”})\rangle \tag{2.3}$$

Os filtros de tuplas estabelecem critérios mais refinados para a recuperação de tuplas. Eles são compostos por um *pattern matching* e uma expressão lógica definida sobre os nomes dos campos das tuplas e possíveis valores que estes podem assumir. Por exemplo, caso a aplicação necessite recuperar apenas leituras da temperatura ambiente cujo valor seja superior a 25 °C, o filtro (2.4) pode ser usado. Observe que a primeira parte do filtro de tupla consiste em um padrão de tupla ((`contextkey`, "`context.ambient.temperature`")) e a segunda parte uma expressão lógica que restringe o limite dos valores da temperatura a serem recuperados (`values` $>$ 25).

$$f = \langle\langle(\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”})\rangle, \quad \texttt{values} > 25\rangle \tag{2.4}$$

A Listagem 1 apresenta um trecho de código que mostra como o filtro de tuplas (*pattern matching* + expressão lógica) descrito em (2.4) pode ser implementado usando a API do *middleware*. No LoCCAM, o padrão de tupla utilizado para as consultas é representado pela classe *Pattern*, em que o desenvolvedor adiciona os campos de interesse na consulta das tuplas. Assim, o SysSU busca no espaço de tuplas todas as tuplas que casam com o padrão informado. No exemplo, o padrão é definido na linha 2 e 3 e é utilizado para consultar as tuplas que contiverem o campo cujo nome é "`ContextKey`" e o valor seja "`context.ambient.temperature`". A interface *IFilter* representa o filtro para o realização dos critérios. No *middleware*, é utilizado um avaliador de expressão, que realiza operações lógicas a fim de verificar se as tuplas que foram recuperadas (i.e., que casaram com o padrão estabelecido) satisfazem a expressão lógica fornecida. Se a expressão for satisfeita, o avaliador retorna "*true*", caso contrário retorna "*false*". No exemplo, o filtro é definido da linha 6 até 14, o campo que será avaliado nessa tupla está na linha 10 ("`Values`") e o valor na linha 11 (maior que "25.0").

2.2.5 Limitações

O LoCCAM foi projetado para manter apenas o estado mais atual dos dados contextuais. Assim, não é possível, por exemplo, que aplicações que necessitam do histórico desses dados sejam implementadas sem a ação do desenvolvedor. Outra limitação encontrada no LoCCAM é que ele foi projetado para ser executado apenas no dispositivo móvel. Assim não é possível que utilizando apenas essa infraestrutura de suporte sensível ao contexto, sejam desenvolvidas aplicações móveis modernas, que apresentam uma tendência na troca de informações (e.g., mensagens).

Listagem 1 – Exemplo de Filtro Contextual

```
1  ...
2  Pattern p = (Pattern) new Pattern();
3  p.addField("ContextKey", "context.ambient.temperature");
4  List<Tuple> result = (ArrayList) loccam.read(p, f);
5  ...
6  IFilter.Stub f = new IFilter.Stub() {
7
8    @Override
9    public boolean filter(Tuple tuple) throws RemoteException {
10   NumberListVariable values = new NumberListVariable("Values", 0);
11   Expression exp = gt(values, new NumberConstant(25.0));
12   return Evaluator.eval(tuple, exp);
13   }
14 }
15 ...
```

As aplicações móveis modernas seguem um modelo de rede social, em que um amplo conjunto de dados heterogêneos (e.g., multimídia) é gerado. Os dados gerados podem ser considerados em uma tomada de decisão para aplicações móveis e sensíveis ao contexto. Outra tendência nessas aplicações é a utilização de dados de outros usuários para melhorar a inferência sobre uma nova situação contextual.

Devido ao aumento da quantidade de dados contextuais utilizados pelas aplicações móveis modernas, o processamento das inferências e gerenciamento de contexto também estão mais complexos. Como mencionado na Seção 2.1.1, os dispositivos móveis são limitados em poder computacional e armazenamento de dados, apesar dos avanços tecnológicos. Assim, realizar essas operações complexas necessárias a essas aplicações, assim como armazenar os dados contextuais que elas utilizam, se torna inviável no dispositivo móvel. Para mitigar tais problemas, buscou-se soluções no campo da *Mobile Cloud Computing* (SHIRAZ *et al.*, 2013).

2.3 Mobile Cloud Computing

Mobile Cloud Computing (MCC) é um paradigma que incorpora duas tecnologias heterogêneas: computação móvel e computação em nuvem (FERNANDO *et al.*, 2013). MCC visa contornar as limitações dos dispositivos móveis em relação ao desempenho e consumo de energia, por meio do uso de recursos da nuvem para aumentar tanto a capacidade de computação

quanto a capacidade de armazenamento desses dispositivos.

Não existe um consenso sobre o conceito de *Mobile Cloud Computing*. Para (SHIRAZ *et al.*, 2013), MCC é “o mais recente paradigma computacional prático, que estende a visão da computação utilitária de computação em nuvem para aumentar os recursos limitados dos dispostivos móveis". Segundo (SANAEI *et al.*, 2014), MCC é “uma rica tecnologia de computação móvel que utiliza recursos elásticos unificados de nuvens variadas e tecnologias de rede com funcionalidade irrestrita, armazenamento e mobilidade para servir uma multidão de dispositivos móveis em qualquer lugar, a qualquer momento através do canal de *Ethernet* ou Internet independentemente de ambientes heterogêneos e plataformas baseadas no princípio *pay-as-you-use*".

2.3.1 Offloading

Existem vários temas de pesquisa relacionados a MCC, sendo destacado na literatura o *offloading* (FERNANDO *et al.*, 2013), técnica que visa aumentar o desempenho e reduzir o consumo de energia de dispositivos móveis por meio da migração de processamento ou dados de dispositivos móveis para outras infraestruturas, com maior poder computacional e armazenamento. É importante destacar que *offloading* é diferente do modelo cliente-servidor tradicional, em que o *thin client* sempre delega a responsabilidade de realizar o processamento ao servidor remoto. No *offloading*, quando não há conexão entre dispositivo móvel e servidor, o processamento é realizado localmente ao dispositivo. Quando existe a conexão e existem ganhos em se migrar dados ou processamento, o *offloading* é realizado.

Tipicamente, existem dois tipos principais de *offloading*: processamento e dados (FERNANDO *et al.*, 2013). O *offloading* de processamento é a entrega de um processamento computacional do dispositivo móvel para outro ambiente de execução (e.g., *laptop*), a fim de prolongar a vida útil da bateria e aumentar a capacidade computacional. O *offloading* de dados tem por objetivo estender a capacidade de armazenamento do dispositivo móvel. Assim, retira-se o armazenamento do dispositivo e enviam-se os dados para um ambiente com maior capacidade de armazenamento, com a possibilidade também de enviar dados processados de volta para o dispositivo móvel.

2.3.2 Questões relacionadas ao Offloading

Existem várias questões de pesquisa relacionadas ao *offloading* (FERNANDO *et al.*, 2013; KUMAR *et al.*, 2013). Dentre as mais importantes, pode-se citar:

- **Onde realizar o *offloading*?**
- **Por que realizar o *offloading*?**
- **Quando realizar o *offloading*?**
- **Fazer o *offloading* de que?**
- **Como executar o *offloading*?**

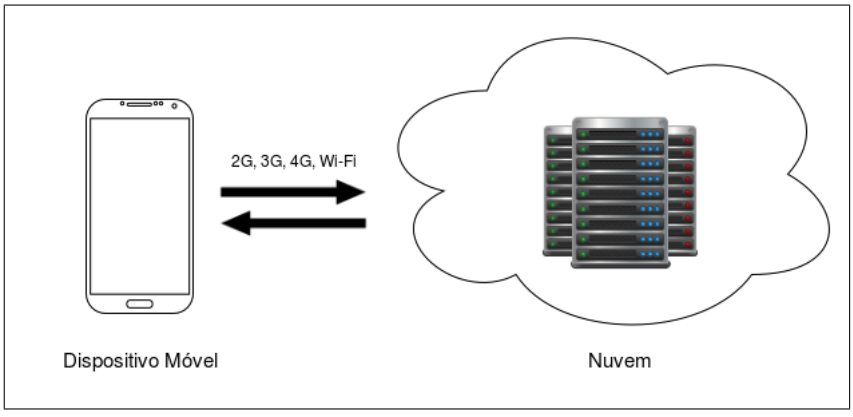
Estas questões são melhor discutidas na seções a seguir.

2.3.2.1 Onde realizar o offloading?

Na técnica de *offloading*, dispositivos móveis usam recursos remotos para melhorar o desempenho da aplicação. Em geral, a execução dessa técnica pode acontecer em três locais possíveis: (*i*) uma nuvem pública; (*ii*) um *cloudlet*; e (*iii*) outro dispositivo móvel.

Na execução em nuvem pública, os desenvolvedores de aplicativos móveis podem utilizar os serviços da nuvem para obter um bom desempenho, assim como elasticidade e conectividade a redes sociais para melhorar os serviços dos aplicativos. Para se conectar à nuvem pública, os dispositivos móveis podem usar redes celulares (e.g., 2G, 3G e 4G) e redes *Wi-Fi*, como ilustrado na Figura 6.

Figura 6 – Nuvem Pública como Ambiente de Execução Remota

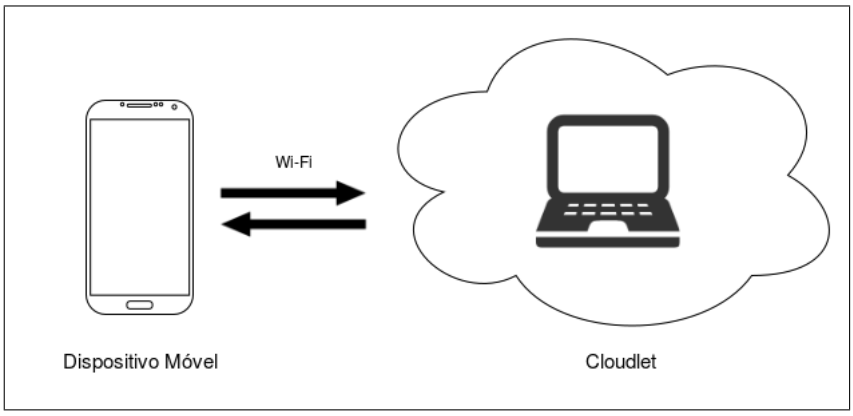


Fonte: Elaborado pelo autor

A execução em *cloudlet* é proposta por Satyanarayanan *et al.* (2009) e tem como principal objetivo aproximar os serviços de *offloading* dos dispositivos móveis para serem executados em máquinas locais, ao invés de utilizar uma nuvem pública. Essas máquinas podem ser *desktops* ou *laptops*, que estão disponíveis em uma mesma rede local sem fio que os usuários das aplicações móveis. O tipo de máquina que compõe um *cloudlet* deve ser proporcional à sua utilidade (COSTA *et al.*, 2015). Por exemplo, uma máquina do porte de um *laptop* talvez seja suficiente para uso pessoal. Em grandes ambientes, como em um aeroporto, é necessário que o *cloudlet* seja composto por uma robusta infraestrutura computacional de servidores, que suportem proporcionalmente as demandas dos usuários.

A vantagem da execução em *cloudlet* é que as redes locais, em geral, possuem velocidades maiores e latência menores em relação aos servidores hospedados em infraestruturas mais remotas, e comumente acessados por redes celulares (COSTA *et al.*, 2015). A Figura 7 ilustra uma visão geral da execução em um *cloudlet*.

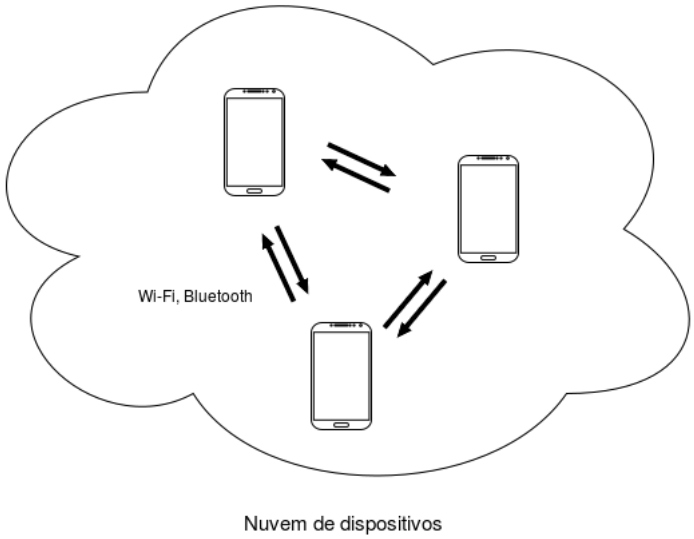
Figura 7 – *Cloudlet* como Ambiente de Execução Remota



Fonte: Elaborado pelo autor

Executar o *offloading* em outro dispositivo móvel com maior poder computacional também é possível. Nessa abordagem, os dispositivos móveis são normalmente conectados usando a rede *peer-to-peer*, criando uma nuvem de dispositivos, em que dispositivos mais robustos cedem parte dos seus recursos computacionais, para resolver uma tarefa em comum a todos os outros dispositivos (FERNANDO *et al.*, 2013) mais fracos. Existem diversas tecnologias que podem ser usadas para a conexão entre os dispositivos (e.g., *Wi-Fi* e *bluetooth*), como é ilustrado na Figura 8.

Figura 8 – Nuvem de Dispositivos como Ambiente de Execução Remota



Fonte: Elaborado pelo autor

O *offloading* também pode ser executado em um ambiente híbrido, que é composto por dois ou mais ambientes mencionados anteriormente.

2.3.2.2 Por que realizar o offloading?

Levando em consideração a limitação de recursos dos dispositivos móveis, o *offloading* de processamento está sendo proposto principalmente para melhorar o desempenho das aplicações móveis e economizar a bateria desses dispositivos. O *offloading* de dados também tem sido proposto para melhorar a colaboração entre usuários das aplicações móveis, assim aumentando a capacidade do armazenamento dos dispositivos móveis.

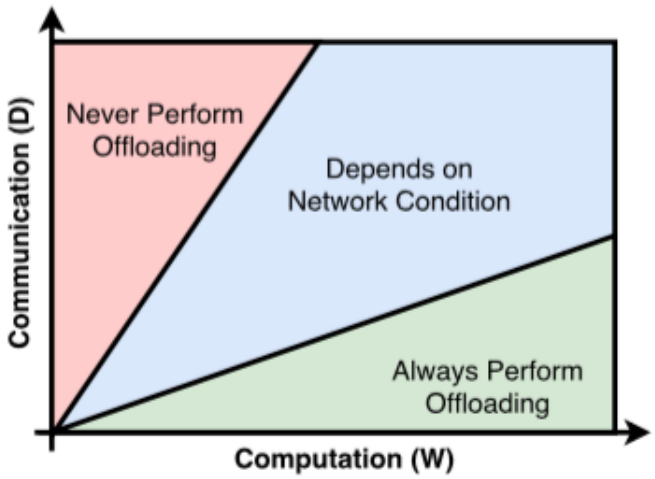
2.3.2.3 Quando realizar o offloading?

Em (KUMAR *et al.*, 2013) é apresentado um modelo analítico para verificar quando o *offloading* melhora o desempenho dos dispositivos móveis. Nesse modelo são comparados o tempo de processamento de uma tarefa no dispositivo móvel e o tempo para enviar, realizar a tarefa em um ambiente remoto e retornar o resultado para o dispositivo móvel. O modelo analítico identifica que para melhorar o desempenho da tarefa é necessário um processamento pesado e uma boa comunicação de rede entre o dispositivo móvel e a nuvem.

A Figura 9 apresenta o *trade-off* entre a computação e a comunicação da rede para

decidir se deve ou não realizar o *offloading*.

Figura 9 – *Trade-off* na Decisão de *Offloading*



Fonte: (KUMAR *et al.*, 2013)

Diferente de Kumar *et al.* (2013), outros pesquisadores focam a decisão se deve ou não realizar o *offloading* de acordo com a economia de energia (KHARBANDA *et al.*, 2012). Rego (2016) propõe uma abordagem que utiliza dados históricos para criar árvores de decisão para auxiliar na tomada de decisão do *offloading*, reduzindo o consumo energético causado pelo monitoramento periódico, mas com todas as operações de computação relacionadas à criação da árvore de decisão executadas em servidores remotos, enquanto os dispositivos móveis só precisam analisá-la. Essa abordagem é uma extensão do modelo de Kumar *et al.* (2013), considerando o tempo de espera no servidor e o tempo para encriptar/decriptar os dados.

2.3.2.4 Fazer o offloading de que?

No *offloading*, é necessário que parte ou toda a aplicação esteja disponível no servidor para que a requisição do cliente seja atendida. Um procedimento de particionamento pode ser utilizado para separar a aplicação em diferentes níveis de granularidade (LIU *et al.*, 2015).

O particionamento da aplicação pode ser realizado automaticamente pela infraestrutura de *offloading* ou transferido a responsabilidade para o desenvolvedor da aplicação que deve adicionar marcações no código (e.g., anotação em Java), para identificar quais componentes são candidatos a serem migrados para o ambiente remoto.

As soluções de *offloading* diferem em relação a que partes da aplicação devem ser

migradas para execução no ambiente remoto. As partes de aplicações mais citadas são: métodos, componentes/módulos, *threads* e aplicação completa. Com relação a migração da aplicação, é importante destacar que as aplicações móveis podem acessar os sensores embarcados no dispositivo. Assim, é inviável realizar o *offloading* nesse caso.

2.3.2.5 Como executar o offloading?

Rego (2016) afirma que não existe uma resposta única e simples para a questão de "Como executar o offloading?". O autor apresenta uma taxonomia para auxiliar na classificação de soluções de *offloading*. Nessa taxonomia, essa questão tem as seguintes categorias:

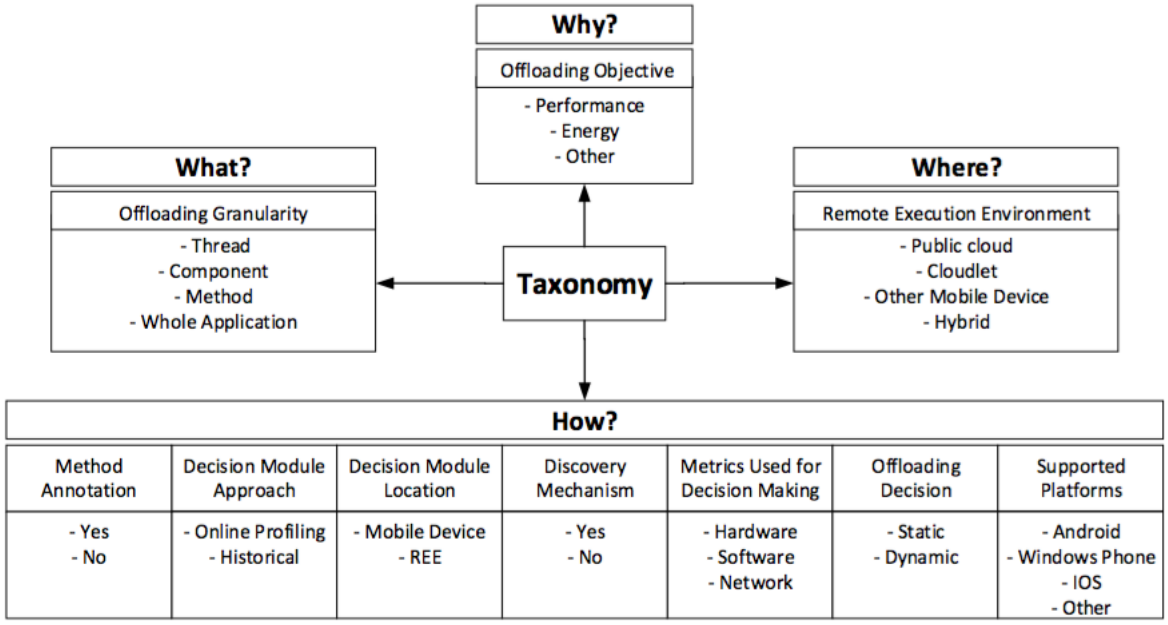
- **Anotação do Método**: quando a solução de *offloading* suporta a granularidade de método;
- **Decisão de *Offloading***: está relacionada com a forma que é feita a decisão de realizar ou não o *offloading*;
- **Localização do Módulo de Decisão**: está relacionada com a localização do módulo responsável pela decisão de realizar o *offloading*;
- **Abordagem do Módulo de Decisão**: está relacionada com as abordagens e técnicas utilizadas pelo módulo de decisão;
- **Métricas Utilizadas para Decisão**: diz respeito às métricas avaliadas no módulo de decisão do *offloading*;
- **Plataforma/Linguagem de Programação Suportada**: diz respeito às plataformas móveis e linguagens de programação suportadas pela solução; e
- **Mecanismo de Descoberta**: essa categoria leva em conta se a solução usa algum tipo de mecanismo de descoberta do ambiente remoto (e.g., *cloudlet*).

A taxonomia proposta em (REGO, 2016) concentra-se em aspectos relacionados ao módulo de decisão de *offloading*. A Figura 10 apresenta essa taxonomia.

2.3.3 Desafios

Como o paradigma de MCC engloba diversas áreas, os desafios técnicos dessas áreas são herdados. A seguir, são listados alguns desses desafios:

Figura 10 – Taxonomia para Soluções de *Offloading*



Fonte: (REGO, 2016)

- **Mobilidade**: Esse é o principal atributo dos dispositivos móveis. É necessário que o acesso aos ambientes remotos seja transparente, imperceptível e ininterrupto (SHIRAZ *et al.*, 2013);
- **Restrições Energéticas**: A tecnologia das baterias não tem acompanhado o avanço tecnológico (SANAEI *et al.*, 2014). Assim, os recursos energéticos são limitados nos dispositivos móveis e se faz necessário que existam meios para conservar a energia (CARROLL; HEISER, 2010);
- **Restrições Computacionais**: Apesar dos avanços tecnológicos, os dispositivos móveis possuem limitações computacionais intrínsecas, referentes às próprias características físicas, como: dimensões, peso, dissipação térmica e consumo elétrico (CARROLL; HEISER, 2010);
- **Privacidade**: Geralmente, os dados do dispositivo móvel são difundidos para a nuvem sem prover nenhuma forma de privacidade para o usuário (SUO *et al.*, 2013). Assim, no caso dos dados conterem dados contextuais do usuário, ele não estaria ciente da possibilidade desses dados terem se tornado públicos.

A seguir é apresentado o *framework* MpOS (COSTA *et al.*, 2015), que é uma infraestrutura que realiza a operação de *offloading* em múltiplas plataformas móveis desenvolvido no laboratório GREat, que assim como o LoCCAM, é um dos pontos de partida do COP.

2.4 MpOS

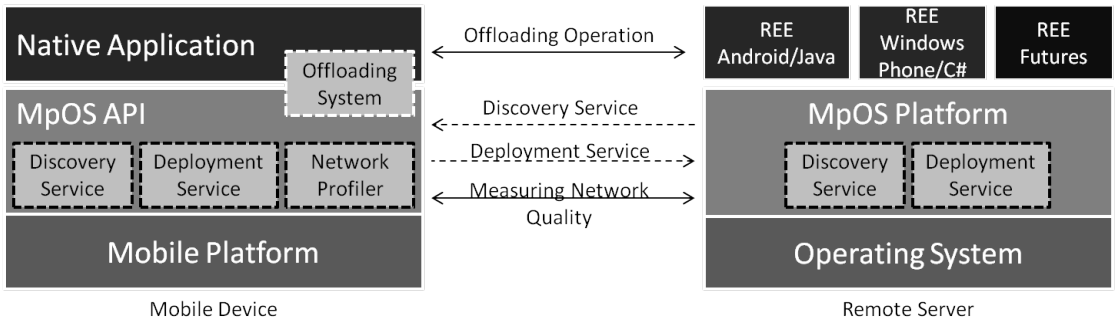
O MpOS é um *framework* para o desenvolvimento de aplicações móveis para múltiplas plataformas móveis (Android e Windows Phone), que realiza a técnica de *offloading* de métodos através de *Remote Procedure Call* (RPC) (COSTA *et al.*, 2015). A aplicação móvel interage com uma biblioteca cliente fornecida pelo *framework* MpOS e possibilita que sejam realizadas as operações de *offloading* para um determinado servidor remoto. O MpOS é composto por serviços que são executados no dispositivo móvel e no servidor remoto. Esses serviços são responsáveis por diversas atividades de redes, como a descoberta e a implantação de serviço, *profiling* de rede, além do ambiente para executar os serviços de *offloading* (COSTA *et al.*, 2015).

Durante o desenvolvimento da aplicação é necessário que o desenvolvedor indique os métodos que podem ser executados em um ambiente remoto, quando for vantajoso. O desenvolvedor também pode escolher se o *offloading* será estático (i.e., sempre será executado) ou dinâmico (i.e., avaliando métricas, por exemplo a latência da rede).

2.4.1 Arquitetura

Uma visão geral da arquitetura do MpOS é apresentada na Figura 11, mostrando as interações entre as camadas. Nessa arquitetura, existem dois tipos de interações de redes. O primeiro tipo são interações unidirecionais, em que acontecem apenas durante a inicialização do aplicativo móvel, como por exemplo, as interações de descoberta de serviço. O segundo tipo são as interações bidirecionais, e que acontecem durante todo o período de execução do aplicativo móvel, como por exemplo, as interações de *offloading* (COSTA *et al.*, 2015).

Figura 11 – Arquitetura do MpOS



Fonte: (COSTA *et al.*, 2015)

A arquitetura baseia-se em uma topologia cliente/servidor, composta de três camadas

que executam tanto no dispositivo móvel, quanto na infraestrutura de nuvem utilizada. A primeira camada do lado cliente representa as aplicações nativas e do lado servidor representa os chamados Ambientes de Execução Remota (AER) ou *Remote Execution Environment* (REE). Esses ambientes são compostos por serviços de rede que processam as operações de *offloading* em diferentes plataformas móveis, como Windows Phone ou Android. Uma vez que o mesmo servidor MpOS pode executar várias REEs, é possível lidar com solicitações de várias aplicações móveis heterogêneas.

A segunda camada do lado cliente possui o MpOS API, enquanto no lado servidor possui o MpOS *Platform*. Os componentes principais dessa camada são:

- **Discovery Service**: o objetivo desse serviço é descobrir um MpOS *Platform* em execução em um *cloudlet* na mesma rede sem fio que o dispositivo móvel. Além disso, esse serviço pode ser usado para obter informações sobre outros serviços do MpOS *Platform* em execução (e.g., *Rede Profiler*). O *Discovery Service* pode ser configurado pelo desenvolvedor para acessar informações em servidores remotos em execução em nuvens públicas, acessando diretamente *endpoints* externos;
- **Deployment Service**: o objetivo desse serviço é implantar a aplicação no REE adequado à sua plataforma de execução. Quando o *Discovery Service* não detecta um serviço de *offloading* no servidor remoto para uma aplicação específica, o *Deployment Service* envia todas as dependências (arquivos binários e bibliotecas) do aplicativo móvel para o servidor. Quando o aplicativo for executado no REE, todos os usuários que estiverem conectados ao MpOS *Platform* podem usar o serviço de *offloading* dessa aplicação, sem necessidade de uma nova implantação;
- **Network Profiler**: o objetivo desse serviço é monitorar a qualidade da conexão entre o dispositivo móvel e o servidor remoto. As métricas usadas para mensurar essa qualidade são: *Round-Trip Time* (RTT), jitter e largura de banda. Estas métricas são utilizadas pelo MpOS para decidir quando realizar o *offloading*; e
- **Offloading System**: o objetivo desse serviço é interceptar os métodos que são marcados pelos desenvolvedores como candidatos ao *offloading*. O *Offloading System* decide onde tais métodos serão executados (i.e., no dispositivo móvel ou no servidor remoto). A decisão é baseada nas métricas monitoradas pelo

Network Profiler e no tipo do particionamento (estático ou dinâmico) empregado na marcação desse método.

A terceira camada MpOS do lado cliente é representada por plataformas móveis (e.g., Android e Windows Phone), enquanto no lado da nuvem tem-se um sistema operacional para desktops ou servidores tradicionais (e.g., Windows).

2.5 Considerações Finais

Este capítulo apresentou os conceitos básicos de Sensibilidade ao Contexto e o paradigma de *Mobile Cloud Computing*. Com relação à Sensibilidade ao Contexto foram apresentadas definições sobre contexto, bem como uma explicação sobre aplicações móveis e sensíveis ao contexto. Também foram apresentadas as principais características de uma infraestrutura de suporte para o desenvolvimento de aplicações sensíveis ao contexto, onde tomou-se como exemplo, o *middleware* LoCCAM. Com relação ao paradigma de *Mobile Cloud Computing* foram apresentado conceitos da área, a técnica de *offloading*, além de algumas das principais questões de pesquisa relacionadas à essa técnica. O *framework* MpOS foi apresentado como exemplo de solução de *offloading*, sendo que o mesmo é utilizado na proposta, assim como o LoCCAM.

O próximo capítulo apresenta os trabalhos relacionados que envolvem soluções que visam o desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizam o compartilhamento de dados contextuais.

3 TRABALHOS RELACIONADOS

Como frisado anteriormente, desenvolver aplicações móveis sensíveis a contexto não é uma tarefa trivial. Para isso, ao longo dos anos foram propostas soluções que buscam facilitar o desenvolvimento dessas aplicações, como *frameworks* e plataformas de *middleware* voltados para o domínio de computação sensível a contexto.

Este capítulo apresenta os principais trabalhos relacionados ao trabalho proposto, que se apresenta também como uma infraestrutura de suporte para facilitar o desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizam o compartilhamento de dados contextuais. A Seção 3.1 apresenta esses trabalhos relacionados. A Seção 3.2 apresenta um comparativo entre esses trabalhos, bem como as características consideradas relevantes durante o estudo deles.

3.1 Infraestruturas de Suporte

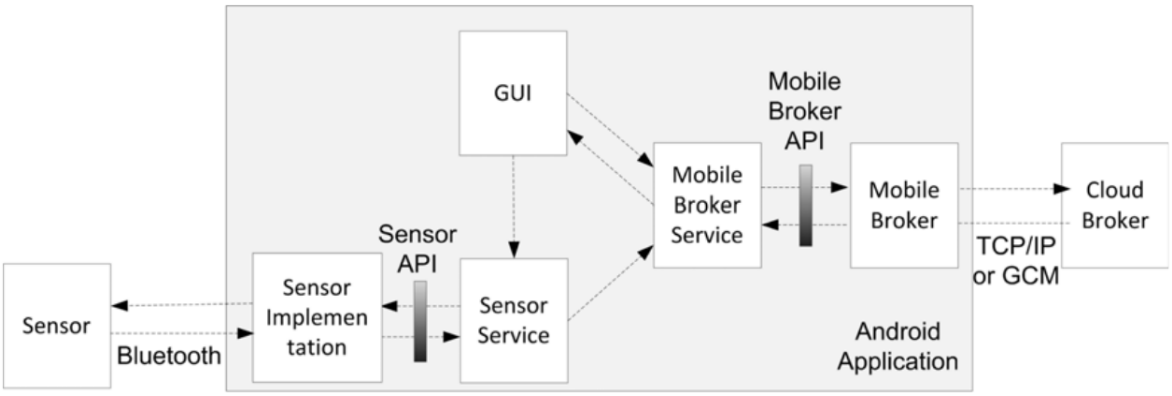
Várias infraestruturas de suporte foram propostas para facilitar o desenvolvimento e aprimorar o reúso e a extensibilidade de aplicações móveis e sensíveis ao contexto (SAEED; WAHEED, 2010; YURUR *et al.*, 2014; KNAPPMEYER *et al.*, 2013). Dentre os trabalhos existentes, foram estudados aqueles que permitem o compartilhamento de dados contextuais. Porém, na seleção dos trabalhos analisados, foram selecionados apenas trabalhos que possuíam pelo menos um dos dois critérios a seguir: (*i*) permitir a configuração da migração dos dados contextuais ou (*ii*) fornecer algum tipo de privacidade para o envio desses dados. Nas próximas subseções são apresentadas as infraestruturas estudadas, bem como suas arquiteturas.

3.1.1 CUPUS

CUPUS (ANTONIć *et al.*, 2016) é um *middleware open-source* baseado em nuvem e adequado para ambientes de Internet das Coisas, especialmente para *deployment* de aplicativos colaborativos. O paradigma de comunicação do CUPUS é *publish-subscribe* e a escalabilidade e elasticidade do *middleware* é facilitada por um design modular, com uma aquisição adaptativa de dados contextuais, assim como no LoCCAM. A arquitetura do CUPUS é ilustrada na Figura 12.

O componente *Mobile Broker* é executado no dispositivo móvel e serve como um *gateway* para os sensores. As principais tarefas desse componente são: (*i*) controlar os sensores

Figura 12 – Arquitetura do CUPUS



Fonte: (ANTONIć *et al.*, 2016)

embarcados e o processo de aquisição de dados contextuais correspondente; (*ii*) pré-processar os dados adquiridos e transmiti-los para a nuvem; e (*iii*) receber as publicações da nuvem e exibir as que são de interesse do usuário.

O componente *Cloud Broker* é utilizado como uma unidade de processamento central dentro da nuvem. As principais tarefas executadas por esse componente são: (*i*) receber e gerenciar as publicações e subscrições recebidas das fontes de dados contextuais e destinos; (*ii*) realizar o *matching* nas subscrições ativas de acordo com as publicações; (*iii*) fornecer as publicações que passaram pelo *matching* a quem se subscreveu; e (*iv*) fornecer as subscrições aos *Mobile Brokers*.

O CUPUS pode parar o processo de aquisição de dados se não houver mais *Mobile Brokers* com interesse nos dados fornecidos, permitindo com isso a economia de recursos. A propriedade mais importante desse *middleware* é a elasticidade, em que o CUPUS implementa um mecanismo de auto-organização de seus componentes de processamento através do ajuste dos recursos da nuvem de acordo com a intensidade da solicitação e a chegada de dados.

O CUPUS foi avaliado em termos de consumo energético, CPU e memória utilizada. Os experimentos mostraram que o consumo energético aumenta com a frequência das publicações dos dados contextuais. Na avaliação da CPU e memória utilizada, cada uma das métricas foi comparada em relação ao número de subscrições e sensores. Os experimentos mostram que conforme aumenta quantidade de subscrições ou de sensores, aumenta o consumo de CPU linearmente. A memória utilizada, independente do aumento da quantidade de subscrições ou de sensores, se mantém constante.

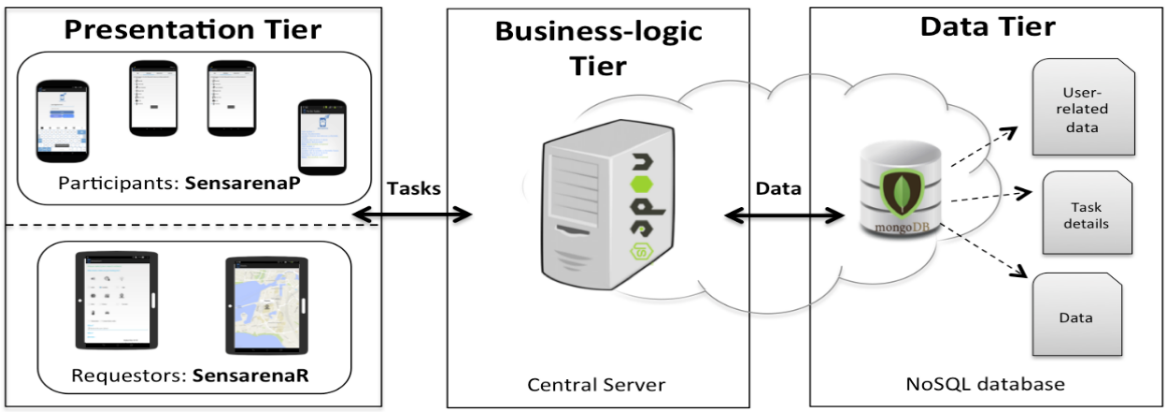
O CUPUS delega às fontes de contexto a forma como os dados contextuais devem ser enviados à nuvem. Entretanto, assim como nos trabalhos anteriores, o CUPUS não define nenhum componente que trate a privacidade dos dados enviados para a nuvem.

3.1.2 Sensarena

Sensarena (MESSAOUD *et al.*, 2016) é um *framework* para facilitar o desenvolvimento de aplicação colaborativas para plataforma Android, que consiste em três principais entidades: dois tipos de aplicações móveis e um servidor central. A primeira aplicação é destinada aos dispositivos móveis que irão realizar a aquisição contextual e a segunda aplicação é para os dispositivos que irão realizar as solicitações de dados contextuais. O servidor do Sensarena possui um mecanismo de atribuição de tarefas de aquisição contextual e armazenamento dos dados.

A arquitetura em camadas do Sensarena é detalhada na Figura 13. A camada de apresentação representa os dispositivos móveis e envolve dois tipos de classes de usuário: os participantes (SensarenaP), que são responsáveis pela aquisição contextual, e os solicitadores (SensarenaR), que são responsáveis pela requisição aos dados contextuais. A camada de lógica de negócios recebe as tarefas de aquisição contextual dos dispositivos móveis e solicita o armazenamento ou recuperação dos dados à camada de dados. Essa última camada está organizada em três coleções principais para armazenar separadamente todas as informações relacionadas ao usuário, os detalhes da tarefa de aquisição contextual e os dados contextuais coletados.

Figura 13 – Arquitetura do Sensarena



Fonte: (MESSAOUD *et al.*, 2016)

A atribuição de tarefas de aquisição contextual do servidor é baseada tanto na posição

geográfica como nas requisições dos usuário. Essa atribuição é executada periodicamente para procurar novas solicitações de aquisição contextual. A primeira atribuição considera todos os participantes registados na área de localização do solicitador. No entanto, o conjunto final de candidatos de uma tarefa específica deve incluir apenas os usuários que atendem aos requisitos da solicitação, sendo esses notificados. Caso nenhum participante da área de localização do solicitador atenda aos requisitos, apenas os participantes adequados do sistema serão notificados sobre os pedidos de detecção disponíveis, especificando previamente a localização e o instante considerado válido.

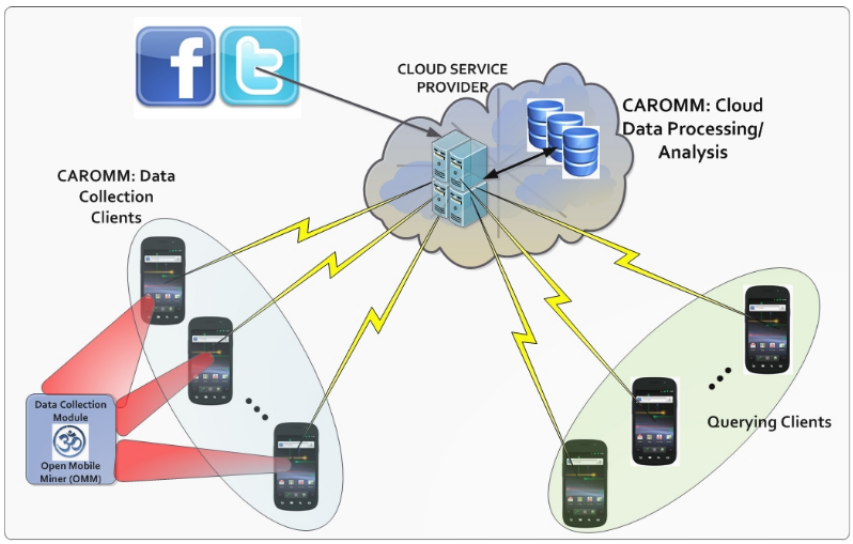
Nesse trabalho é fornecido aos usuários um mecanismo de privacidade capaz de filtrar os dados dos sensores que serão compartilhados, mas não possui nenhuma política de sincronização de dados contextuais do dispositivo móvel para o ambiente remoto. Os autores não realizaram nenhuma avaliação da solução.

3.1.3 CAROMM

CAROMM (SHERCHAN *et al.*, 2012) é um *framework* para aplicações móveis para plataforma Android que busca utilizar dados provenientes de muitos usuários. Essa infraestrutura visa facilitar a coleta de dados do sensores oriundos dos dispositivos móveis e correlacionar esses dados com dados de redes sociais em tempo real. O CAROMM foi projetado com base nos seguintes princípios: capturar diferentes tipos de *stream* de dados oriundos dos dispositivos móveis; processar, gerenciar e analisar tais dados com os dados contextuais associados (e.g, associar a intensidade da luz com fotos); e facilitar as consultas em tempo real que venham dos dispositivos. O *framework* consiste de três módulos principais (Figura 14): *Data Collection Client* e *Querying Client*, que ficam no dispositivo móvel, e o *Data Processing Module*, que fica na nuvem.

O *Data Collection Module* captura os dados oriundos dos sensores, realiza uma mineração contínua em tempo real dos dados e faz o *upload* para a nuvem dos dados analisados para o *Data Processing Module*. Na nuvem, os dados são re-analisados, gerenciados e acontece a fusão dos múltiplos fluxos de *stream*. Para o envio inteligente dos dados, é utilizado um *resource-aware clustering* sobre os dados sensoreados para identificar mudanças significativas do ambiente. Isso reduz a frequência e a grande quantidade de dados transferidos dos dispositivos móveis para a nuvem, enquanto que garante que dados importantes não sejam perdidos. O

Figura 14 – Visão geral do CAROMM



Fonte: (SHERCHAN *et al.*, 2012)

Querying Client no dispositvo móvel envia as consultas dos usuários para o *Data Processing Module*, que recebe os pedidos e devolve os resultados obtidos.

O *Data Processing Module* consiste de dois componentes: *Social Media Data Collection* e *Query Processing*. A principal função desse módulo é agregar os dados obtidos de todas as fontes de dados e entregar o dado contextual referente às consultas dos usuários.

O *framework* executa em um dispositivo móvel compatível com a plataforma Android e possibilita o ajuste de parâmetros (e.g, frequência de envio dos dados e habilitar sensores). O lado nuvem utiliza os serviços da Amazon AWS[1]. Assim como os outros trabalhos, a privacidade dos dados contextuais é negligenciada.

3.1.4 Sahyog

Sahyog é um *middleware* proposto por Bajaj e Singh (2015) e desenvolvido para plataforma Android, cujo objetivo é fornecer o suporte ao desenvolvimento de aplicações colaborativas. A seguir são listadas algumas características estabelecidas para o *middleware*:

- os usuários podem optar por compartilhar ou não os seus dados com os outros usuários;
- pode ser fornecido aos usuários o controle completo sobre dados que os mesmos

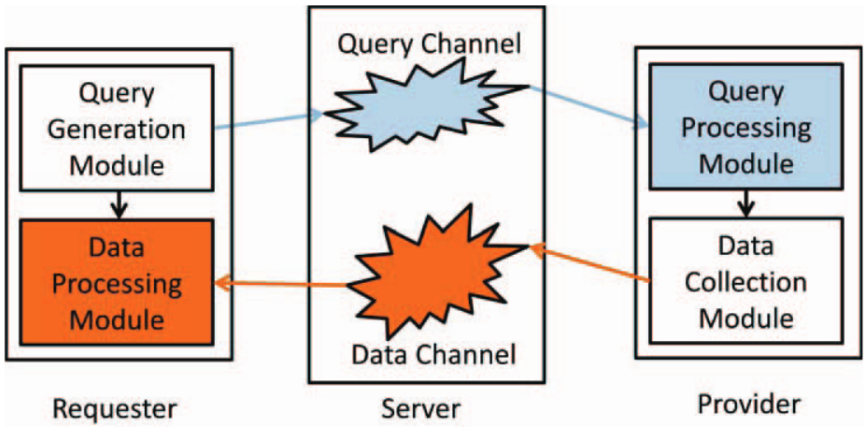
[1] https://aws.amazon.com/

desejam compartilhar, ao mesmo tempo que garante-se o anonimato do usuário; e

- é oferecido o suporte à colaboração em tempo real.

Sahyog utiliza o tipo de comunicação *publish-subscribe*, e sua arquitetura básica pode ser vista na Figura 15. Nela existem dois componentes básicos: dispositivo móvel (clientes) e o *PubSub framework* (servidor). O cliente pode ser tanto um *Requester* como um *Provider*. O *Request* publica uma consulta para um dado do interesse dele e o *Provider* responde a consulta de outros usuários com os dados solicitados. No servidor existem dois tipos de canais que distinguem a consulta dos dados e das requisições.

Figura 15 – Arquitetura do Sahyog



Fonte: (BAJAJ; SINGH, 2015)

Sahyog utiliza mensagens *JavaScript Object Notation* (JSON)[2] na troca de dados contextuais, o que permite uma flexibilidade para as consultas. Na representação desses dados, existem parâmetros obrigatórios que podem invalidar um dado e assim não retorná-lo para os usuários interessados (e.g., *timestamp*), sendo uma deficiência desse trabalho. No Sahyog não é permitido um ajuste do envio dos dados contextuais do dispositivo móvel para a nuvem.

3.2 Comparativo Entre os Trabalhos

Para o comparativo entre os trabalhos relacionados, foram levantadas características relevantes desses trabalhos em relação ao serviço proposto nesta dissertação. As características consideradas para a construção de uma infraestrutura de suporte que visa facilitar o desenvolvi-

[2] http://www.json.org/

mento de aplicações móveis e sensíveis ao contexto que utilizam o compartilhamento de dados contextuais são as seguintes:

- **Tipo**: essa característica diz respeito ao tipo de infraestrutura de suporte da solução (e.g., *framework*, *middleware*);
- **Plataforma**: refere-se às plataformas móveis suportadas pela solução (e.g., Android, iOS, Windows Phone);
- **Paradigma de Comunicação**: corresponde ao tipo de comunicação entre as aplicações e a solução (e.g., *Request-Response* e *Publish-Subscribe*);
- **Modelo de Contexto**: refere-se ao modelo de contexto utilizado na solução (e.g., chave-valor, ontologia);
- **Política de Sincronização**: essa característica refere-se à possibilidade de configurar como os dados contextuais serão enviados para o ambiente remoto na solução; e
- **Privacidade**: essa característica mostra se a solução possui algum mecanismo de privacidade para os dados contextuais dos usuários.

A Tabela 1 detalha o comparativo entre os trabalhos relacionados e as características elencadas.

Tabela 1 – Comparativo Entre os Trabalhos Relacionados

Trabalho	Tipo	Plataforma	Paradigma de Comunicação	Modelo de Contexto	Política de Sincronização	Privacidade
CUPUS	*Middleware*	Android	*Publish-Subscribe*	Chave-Valor	Sim	Não
Sensarena	*Framework*	Android	*Request-Response*	Chave-Valor	Não	Sim
CAROMM	*Framework*	Android	*Request-Response*	Linguagem de Marcação	Sim	Não
Sahyog	*Middleware*	Android	*Request-Response* *Publish-Subscribe*	Chave-Valor	Não	Sim

Fonte: Elaborado pelo autor

3.3 Considerações Finais

Esse capítulo apresentou os trabalhos relacionados e um conjunto de características elencadas para uma infraestrutura de suporte sensível ao contexto, com foco no compartilhamento de dados contextuais.

De acordo com a revisão da literatura, foram encontradas quatro trabalhos que mais

se relacionam com o trabalho proposto nesta dissertação, e elencadas as seguintes características para a comparação entre essas soluções: Tipo, Plataforma, Paradigma de Comunicação, Modelo de Contexto, Políticas de Sincronização e Privacidade.

Depois de realizada a comparação entre as características e as soluções encontradas, foi possível verificar que nenhuma das soluções consegue prover ao usuário do sistema, um mecanismo configurável de envio e de privacidade dos dados contextuais do dispositivo móvel para o ambiente remoto. Dessa forma, o próximo capítulo apresenta a solução proposta, sua arquitetura, seu modelo de contexto, seu modelo de privacidade e as políticas de sincronização.

4 SERVIÇO DE *OFFLOADING* DE DADOS CONTEXTUAIS COM SUPORTE À PRIVACIDADE

Este capítulo apresenta o trabalho proposto. Uma visão geral do trabalho é apresentada na Seção 4.1. A Seção 4.2 apresenta a arquitetura da solução. A Seção 4.3 explica como um dado contextual é representado pelo trabalho. A política de privacidade para restringir o envio dos dados contextuais do dispositivo móvel para um ambiente remoto é explicada na Seção 4.4. Por fim, a Seção 4.5 apresenta as políticas de sincronização para o envio desses dados.

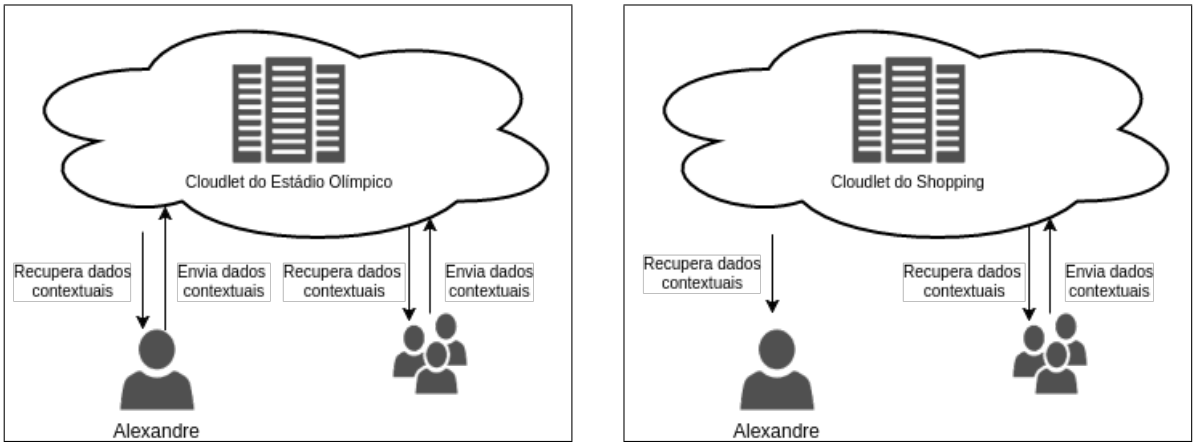
4.1 Visão Geral

A solução proposta é uma infraestrutura de suporte para facilitar o desenvolvimento de aplicações móveis e sensíveis ao contexto para plataforma Android, com suporte à disseminação de dados contextuais e privacidade. A solução proposta segue o modelo de um serviço batizado de COP (***C****ontextual data* ***O****ffloading service with* ***P****rivacy support*), e baseia-se em: *(i)* um modelo de contexto; *(ii)* uma política de privacidade; e *(iii)* políticas de sincronização. Para contextualizar melhor a relevância do trabalho proposto, apresenta-se a seguir um cenário motivador para sua utilização.

Alexandre está na abertura dos Jogos Olímpicos do Rio 2016. Alexandre e milhares de usuários estão publicando fotos e marcando-as com *hashtags* por meio de um aplicativo móvel. O aplicativo sugere *hashtags* de fotos com contexto semelhante (i.e., com local e instante semelhantes ao de novas fotos). Sempre que possível, a aplicação realiza o *offloading* de dados no *cloudlet* que executa no Estádio Olímpico. Quando Alexandre captura uma foto, são recomendadas as cinco *hashtags* mais utilizadas entre os usuários do aplicativo durante a festa de abertura. Depois do jogo, Alexandre vai a um *shopping* na cidade que também possui um *cloudlet*. Ele continua utilizando o aplicativo. Por receio que outros usuários saibam a sua localização, Alexandre desativa o envio dos seus dados contextuais, dentre eles, a sua localização.

A Figura 16 ilustra esse cenário, o qual é composto de dois momentos distintos. No primeiro momento (Figura 16(a)), Alexandre tanto compartilha as *hashtags* dele como recebe recomendações sobre as *hashtags* mais comentadas na hora da cerimônia de abertura. No segundo momento (Figura 16(b)), Alexandre não quer tornar pública a sua localização, e apenas recebe os dados contextuais de outros usuários, porém sem enviar seus dados.

Figura 16 – Cenário Motivador



(a) *Cloudlet* Confiável para o Usuário (b) *Cloudlet* não Confiável para o Usuário

Fonte: Elaborado pelo autor

A partir da visão geral do cenário desejado, uma série de requisitos foi encontrada. Primeiro, uma vez que dados contextuais de diferentes usuários e dispositivos são agregados em um *cloudlet/cloud*, faz-se necessário elaborar um modelo de como estes dados seriam representados. Esta representação deve permitir tanto que o usuário possa agregar dados históricos de seus próprios dados contextuais, quanto o uso de dados contextuais de diferentes usuários, quando desejado.

De modo a tratar a privacidade, também faz-se necessário elaborar um esquema que controla a divulgação dos dados contextuais dos usuários. Além deste controle, a forma como os dados são disseminados entre dispositivos móveis e o ambiente de nuvem também merece atenção, de modo a não sobrecarregar a rede ou mesmo exaurir os recursos dos dispositivos móveis.

Para tratar as questões apresentadas, esta dissertação propõe uma extensão do *middleware* LoCCAM, integrando os dados contextuais originalmente restritos a um único dispositivo móvel a uma estrutura de nuvem configurável (*cloud/cloudlet*). O modelo de representação contextual proposto pelo LoCCAM foi estendido, de modo a permitir que dados de mais de um dispositivo (usuário) possam ser agregados em um espaço de tuplas compartilhado. Desta forma, é possível que determinados filtros contextuais, que outrora seriam executados apenas no dispositivo móvel, possam ser executados na nuvem. Para a execução desses filtros na nuvem foi utilizado o *framework* MpOS, que realiza operações de *offloading*, a fim de oferecer um suporte à melhoria da caracterização de uma dada situação contextual.

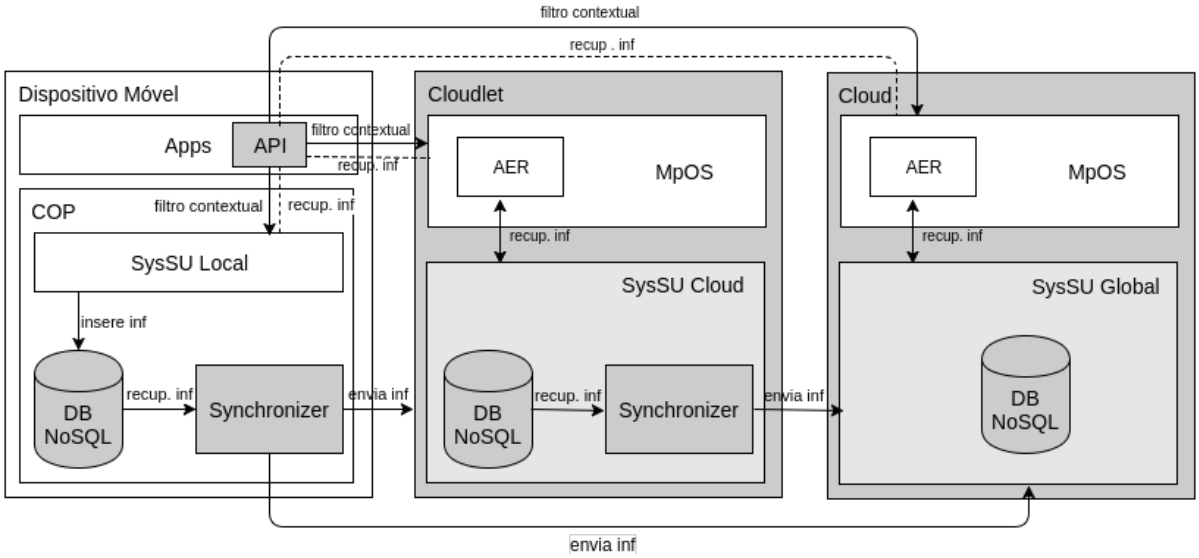
Além disso, é proposto um conjunto de regras que indicam como os dados contextuais são enviados pelos dispositivos móveis para a nuvem. Estas regras são chamadas nesta dissertação de políticas de sincronização. Por fim, foi especificado o serviço do COP para que o usuário possa explicitamente indicar quais políticas de sincronização ele deseja utilizar para o envio dos dados contextuais para a nuvem, e quais desses dados devem ser compartilhados. As seções a seguir explicam cada um destes aspectos.

4.2 Arquitetura

Para solucionar as necessidades encontradas no cenário motivador, foi definida uma arquitetura composta por 3 camadas, que é apresentada na Figura 17. A primeira camada é representada pelo dispositivo móvel, onde os dados contextuais são capturados e armazenados localmente. A segunda camada é representada por um cloudlet, que recebe dados contextuais de usuários conectados a uma mesma *Wireless Local Network* (WLAN). Por fim, o último nível representa um espaço de tuplas hospedado em um serviço de nuvem, e que pode agregar dados de diferentes *cloudlets* e usuários que compartilham seus dados contextuais.

No lado do dispositivo móvel, foi acrescentado um banco de dados NoSQL, que atua como uma espécie de *cache* local das informações de contexto, até que o processo de *offloading* de dados ocorra.

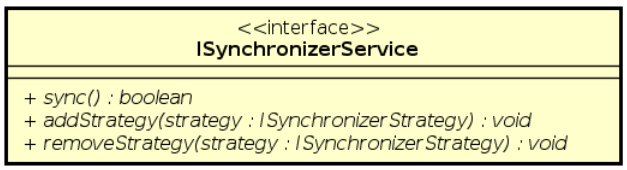
Figura 17 – Arquitetura do COP



Fonte: Elaborado pelo autor

O componente *Synchronizer* presente no dispositivo móvel é responsável por recuperar os dados contextuais do banco de dados no dispositivo móvel e enviar para o ambiente remoto. O *Synchronizer* utiliza de estratégias de envio definidas nas políticas de sincronização (mais detalhes na Seção 4.5). Sempre que as tuplas forem enviadas para o ambiente remoto, o banco de dados local é esvaziado para evitar o acúmulo de dados no dispositivo móvel. O *Synchronizer* possui uma interface que possibilita a implementação de um algoritmo de sincronização diferente, caso necessário (ver Figura 18).

Figura 18 – Interface Implementada pelo *Synchronizer*



Fonte: Elaborado pelo autor

O método *sync* da Interface *Syncronizer* é responsável pelo envio de dados contextuais do dispositivo móvel para o ambiente remoto. Os outros dois métodos recebem como parâmetro uma estratégia de envio, que deve implementar a Interface *ISyncronizerStrategy* (mais detalhes na Seção 4.5). O método *addStrategy* tem a finalidade de adicionar cada estratégia de envio que será utilizada pelo serviço proposto. O método *removeStrategy* tem o objetivo de remover estratégias que não sejam mais de interesse do usuário do serviço.

Tanto no *cloudlet* como na nuvem, existem componentes responsáveis pelo gerenciamento de dados contextuais compartilhados. No caso do *cloudlet*, este componente é chamado de SysSU *Cloud*, enquanto que o serviço que agrega dados de todos os *cloudlets* e usuários é chamado de SysSU *Global*. Ambos possuem um banco de dados NoSQL, similar ao que executa no dispositivo móvel, e que tem por objetivo a persistência dos dados contextuais dos dispositivos móveis que compõem o sistema.

No COP, além das tuplas dos dispositivos móveis serem enviadas para o *cloudlet*, elas podem ser enviadas entre *cloudlets*, ou para um repositório central na nuvem, permitindo que os dados contextuais de diversos dispositivos móveis sejam aglutinados em um único local. Diferente do lado móvel, não existe uma política de esvaziamento do banco de dados no *cloudlet*, pois se caso a nuvem seja inacessível (i.e., problemas de conexão), basta que o dispositivo

móvel possua conexão *Wi-Fi* e um *cloudlet* em execução, para que o usuário acesse os dados compartilhados.

Para o envio dos dados contextuais dos *cloudlets* para a nuvem, o SysSU *Cloud* possui um componente *Synchronizer* que utiliza a estratégia de envio por tempo. É importante destacar que caso não seja descoberto um *cloudlet* em execução, os dados contextuais podem ser enviados diretamente para o SysSU *Global*.

Filtros Contextuais no COP

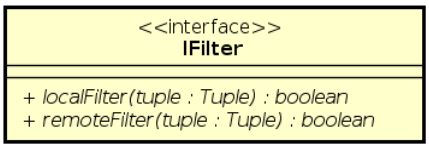
Assim como no LoCCAM, o COP acessa os dados contextuais por meio de filtros contextuais de forma transparente, podendo ser no dispositivo móvel (SysSU *Local*), no *cloudlet* (SysSU *Cloud*) ou na nuvem (SysSU *Global*). Para isso, é utilizado o *framework* MpOS, apresentado na Seção 2.4.

Dentre outras funcionalidades, o MpOS decide se o *offloading* deve ser executado ou não no ambiente remoto, a fim de melhorar o desempenho da aplicação. Para a tomada de decisão, são usadas métricas de rede, especialmente em relação à qualidade da conexão entre o dispositivo móvel e o servidor remoto (e.g., latência da conexão).

A partir da utilização do MpOS, a API do COP realiza a descoberta do local de execução da aplicação e direciona a execução dos filtros contextuais sobre as tuplas. Se o local de execução for o dispositivo móvel, esse filtro será executado sobre as tuplas que estão no SysSU *Local*. Se esse local for um *cloudlet* ou uma nuvem, quando oportuno, esses filtros são descarregados para esses locais, especificamente no componente Ambiente de Execução Remota (AER) do MpOS, e executados sobre as informações de contexto que estão no SysSU *Cloud* ou SysSU *Global*, respectivamente.

Para a execução dos filtros contextuais tanto no dispositivo móvel como nos ambientes remotos, é necessário que o desenvolvedor da aplicação implemente tais filtros. Esses filtros podem ser diferentes, pois os algoritmos remotos podem ser mais complexos que os algoritmos locais, uma vez que são executados em locais com maior capacidade de processamento e podem ter que lidar com maior quantidade de dados. A interface *IFilter* presente no LoCCAM foi modificada para a implementação desses filtros (ver Figura 19).

Figura 19 – Interface do Filtro Contextual no COP



Fonte: Elaborado pelo autor

De acordo com a especificação de um filtro, o desenvolvedor deve dar corpo ao método *localFilter*, a ser executado no dispositivo móvel, e ao método *remoteFilter*, a ser executado no ambiente remoto, quando oportuno.

4.3 Representação do Modelo de Contexto

Com relação ao modelo de contexto utilizado no trabalho proposto, foi realizado a extensão do modelo de contexto definido para o LoCCAM. Esta extensão exigiu modificações tanto no tipo, quanto nos dados representados pelas tuplas. No COP, os dados contextuais de múltiplos usuários são armazenadas em um espaço de tuplas compartilhado. Com isso são necessárias novas informações para individualizar os dados, o que indica que as tuplas no COP possuem mais campos para garantir a identificação da origem dos dados de contexto, lidar com questões de relógio em ambiente distribuído e com a privacidade dos dados, conforme Definição 3.

Definição 3 (Tupla no COP) *Uma tupla no COP é representada por nove elementos, sendo cinco herdados da Definição 2 e quatro novos necessários, são eles:* $t = (\texttt{iddevice}, \text{“?”}), (\texttt{idapp}, \text{“?”}), (\texttt{visible}, \text{“?”}), (\texttt{globaltimestamp}, \text{“?”})\rangle$*, onde* `iddevice` *representa a identificação do dispositivo móvel;* `idapp` *representa a identificação da aplicação que está utilizando o middleware;* `visible` *é utilizado para identificar a visibilidade do dado (mais detalhes na Seção 4.4); e* `globaltimestamp` *representa o instante, em milissegundos, em que o dado contextual é inserido no ambiente remoto.*

A estrutura apresentada em (4.1) caracteriza uma leitura do dado de contexto “temperatura do ambiente” no formato de tupla para o COP. Os campos sublinhados foram adicionados aos já existentes. Nessa estrutura, o dispositivo móvel tem identificação “0424033418” e utiliza

uma aplicação com identificação "br.casa.temp".

$$
\begin{aligned}
t = \langle & (\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”}), \\
& (\texttt{source}, \text{“}\texttt{physical}\text{”}), \\
& (\texttt{values}, \text{“}25\text{”}), \\
& (\texttt{accuracy}, \text{“}0\text{”}), \\
& (\texttt{timestamp}, \text{“}239459060969\text{”}) \\
& (\underline{\texttt{iddevice}}, \text{“}0424033418\text{”}) \\
& (\underline{\texttt{idapp}}, \text{“}\texttt{br.casa.temp}\text{”}) \\
& (\underline{\texttt{visible}}, \text{“}0\text{”}) \\
& (\underline{\texttt{globaltimestamp}}, \text{“}239459780932\text{”})\rangle
\end{aligned}
\tag{4.1}
$$

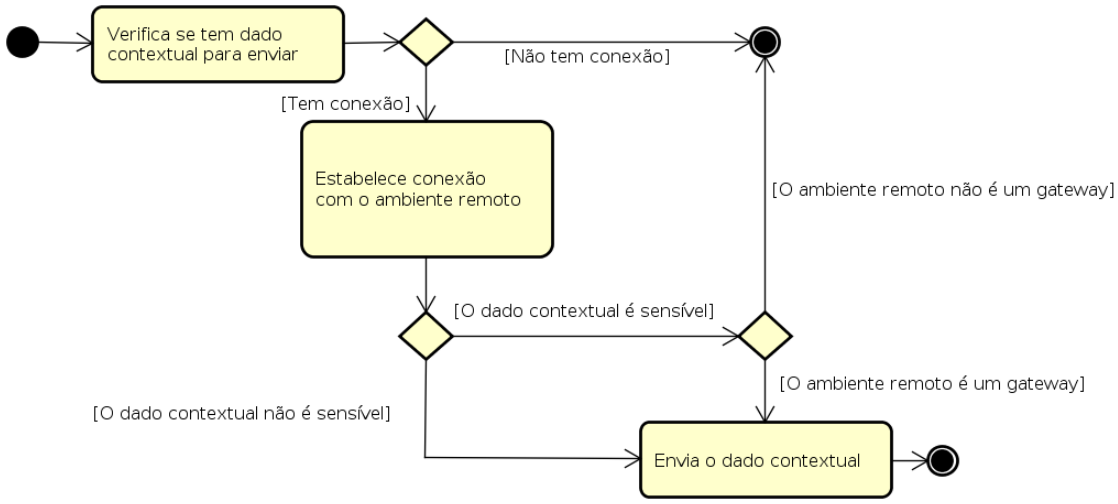
A representação de contexto no COP acrescentou mais informações à representação que foi estendida, consequentemente aumentou a necessidade de mais memória para armazenar os dados contextuais. Porém, como o COP migra os dados de cada dispositivo móvel para um ambiente remoto, o aumento da representação de contexto não chega a ser uma limitação da solução.

4.4 Política de Privacidade

Com relação à política de privacidade proposta no trabalho, foram estabelecidos dois conceitos: dados contextuais sensíveis e *cloudlet* confiável. Os dados contextuais sensíveis são aqueles considerados sigilosos (HENGARTNER; STEENKISTE, 2006). No COP, o usuário pode estabelecer explicitamente quais dados ele autoriza (ou não) serem enviados para o ambiente remoto. A privacidade dos dados contextuais está relacionada com a visibilidade dos mesmos, que é representada pelo campo *visible* da tupla do COP (ver Definição 3). O valor desse campo deve receber o valor "1" para indicar que a visibilidade é pública, ou "0" caso a visibilidade seja privada. Na estrutura apresentada em (4.1), o valor do campo `visible` é "0". Assim, esse dado contextual é dito sensível e não será enviado para o ambiente remoto. Na prática, o usuário realiza a definição dos dados contextuais sensíveis através do serviço do COP, que executa no dispositivo móvel, selecionando quais os dados contextuais são privados, a partir das *ContextKeys* disponíveis (ver Apêndice A).

O *cloudlet* confiável ou *gateway* é definido pelo usuário e seria, por exemplo, um *desktop* na residência desse usuário. Se os dados contextuais forem definidos como privados, eles poderão ser difundidos para um *cloudlet*, desde que este seja considerado como confiável pelo usuário. Contudo, os dados privados armazenados nos *cloudlets* confiáveis não são enviados para o SysSU *Global*. Na prática, o usuário realiza a definição do *gateway* através do serviço do COP, que executa no dispositivo móvel, especificando o *endpoint* do *cloudlet* (ver Apêndice A). Um diagrama da política de privacidade do COP é ilustrado na Figura 20.

Figura 20 – Diagrama da Política de Privacidade



Fonte: Elaborado pelo autor

Assim, na política de privacidade do COP, no momento da migração dos dados se o dispositivo móvel não estiver conectado a um *cloudlet* confiável, apenas as tuplas públicas serão enviadas. Caso contrário, se o dispositivo estiver conectado a um *cloudlet* confiável todos os dados contextuais serão enviados, independente da sua visibilidade.

4.5 Políticas de Sincronização

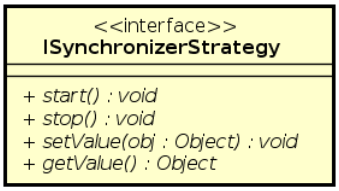
Um outro ponto importante é definir em que momentos os dados contextuais devem ser migrados do dispositivo para um ambiente remoto. Neste trabalho foram estabelecidas estratégias diferentes de envio dos dados contextuais do dispositivo móvel para o ambiente remoto, as quais são denominadas de políticas de sincronização.

Atualmente três estratégias de envio foram especificadas, a saber: (*i*) por tempo, (*ii*) por quantidade de tuplas e (*iii*) orientada à conexão *Wi-Fi*. A estratégia de envio por

tempo significa que, periodicamente (e.g., a cada 30 segundos), as tuplas são enviadas para o ambiente remoto. O período para o envio dos dados é configurável. Caso o usuário não defina explicitamente outra estratégia a ser usada, esta é a estratégia padrão utilizada. A estratégia de envio por quantidade de tuplas implica que, se uma quantidade pré-estabelecida de tuplas for alcançada, as tuplas serão enviadas à nuvem. Essa quantidade também é configurável. Por fim, a estratégia de envio orientada à conexão *Wi-Fi* define que apenas quando uma conexão *Wi-Fi* for estabelecida, as tuplas do dispositivo móvel serão enviadas à infraestrutura de nuvem. Se caso o usuário estabeleça essa política, mas utilize uma conexão 3G/4G, os dados contextuais não serão descarregados.

A Figura 21 apresenta a Interface comum implementada por toda estratégia no COP. Essa Interface, garante que certos métodos sejam implementados e disponibilizados publicamente. Assim, o trabalho proposto pode ser estendido, caso novas políticas de sincronização venham a ser estabelecidas.

Figura 21 – Interface Implementada pelas Estratégias



Fonte: Elaborado pelo autor

O método *start* dessa Interface é responsável por iniciar a execução da estratégia de envio dos dados contextuais do dispositivo móvel para o ambiente remoto. Quando o usuário do sistema desejar parar de enviar seus dados, o método *stop* é invocado. O método *setValue* é responsável por configurar o limiar da estratégia (e.g., se uma estratégia por tempo for projetada para migrar dados a cada 30 segundos, então o limiar tem valor 30). O método *getValue* tem a função de informar o valor desse limiar.

Essas estratégias podem ser combinadas quando escolhidas simultaneamente pelo usuário. Neste caso, a que ocorrer primeiro será executada. Por exemplo, se um usuário define que deseja utilizar as estratégias por tempo e por quantidade de tuplas, quando a quantidade de tuplas pré-estabelecidas for atingida antes do tempo definido, essas tuplas serão enviadas. O mesmo acontece se a estratégia por tempo acontecer antes. Na prática, o usuário realiza a definição das

estratégias através do serviço do COP, que executa no dispositivo móvel, selecionando quais estratégias ele gostaria de utilizar a partir da seleção das opções existentes, e definindo um limiar para o serviço (ver Apêndice A).

4.6 Considerações Finais

Neste capítulo foi apresentada a proposta de um serviço de *offloading* de dados contextuais com suporte a privacidade chamado de COP (***C****ontextual data* ***O****ffloading service with* ***P****rivacy Support*), que é uma extensão do *middleware* LoCCAM, e baseia-se em: (*i*) um modelo de contexto; (*ii*) uma política de privacidade; e (*iii*) políticas de sincronização.

Essa solução busca mitigar o problema do armazenamento de dados nos dispositivos móveis com o uso da técnica de *offloading*, que consiste na migração de processamento ou dados para um ambiente com a capacidade de realizar essas tarefas. Para o envio desses dados, foram definidas políticas de sincronização. Entretanto, quando os dados do dispositivo móvel são difundidos para a nuvem, é necessário que exista algum mecanismo de controle de privacidade para o usuário, pois quando a privacidade é negligenciada, pode acarretar na divulgação pública dos dados contextuais do usuário sem o seu devido consentimento. Uma solução para essa questão foi a utilização de uma política de privacidade no COP.

O próximo capítulo apresenta exemplos de códigos de utilização do serviço proposto através da implementação de uma aplicação como prova de conceito, que baseia-se no cenário motivador. O próximo capítulo também será avaliado o impacto do processamento dos filtros contextuais tanto no dispositivo móvel como em um *cloudlet* e o consumo energético das políticas de sincronização. Por fim, será apresentado um comparativo entre a solução proposta e os trabalhos relacionados.

5 PROVA DE CONCEITO E AVALIAÇÕES

Como forma de ilustrar os ganhos da migração de dados dos dispositivos móveis que utilizam esta solução, foi implementado como prova de conceito um aplicativo de recomendação intitulado MyPhotos. Este aplicativo baseia-se no cenário motivador apresentado previamente no capítulo 4. A implementação dessa aplicação é demonstrada passo a passo. A aplicação MyPhotos demonstra como é possível utilizar o serviço do COP para o desenvolvimento de aplicações móveis e sensíveis ao contexto que compartilham os dados contextuais dos usuários do sistema, bem como a necessidade de controlar a privacidade desses dados durante a migração do dispositivo móvel para um ambiente remoto para que não ocorra a divulgação sem a devida permissão.

Para avaliar o impacto do processamento dos filtros contextuais na recuperação dos dados contextuais tanto no dispositivo móvel como em um ambiente remoto foi desenvolvida uma segunda aplicação. Essa aplicação foi utilizada para realização de testes que avaliaram o tempo de processamento e o gasto energético desses filtros no dispositivo móvel e em um *cloudlet*.

Para avaliar o impacto das políticas de sincronização implementadas para o envio dos dados contextuais do dispositivo móvel para um ambiente remoto, foi desenvolvido uma terceira aplicação. Essa aplicação foi utilizada para realização de um teste que avaliou o gasto energético dessas políticas no dispositivo móvel durante o processo de envio dos dados contextuais para um *cloudlet*.

A Seção 5.1 apresenta a prova de conceito e como ela foi implementada passo a passo, bem como um experimento realizado e os resultados obtidos com ele. Já a Seção 5.2 expõe a segunda aplicação, chamada *Integers*, sobre a qual ocorreu a realização de testes de tempo de processamento e gasto energético dos filtros contextuais no dispositivo móvel e em um *cloudlet* e os resultados obtidos. A Seção 5.3 apresenta a terceira aplicação, chamada *Sending*, sobre a qual ocorreu a realização do teste de gasto energético das políticas de sincronização no dispositivo móvel durante o processo de envio dos dados contextuais para um *cloudlet* e os resultados obtidos. Por fim, um comparativo entre a solução proposta e os trabalhos relacionados é apresentado na Seção 5.4

5.1 Prova de Conceito

MyPhotos é uma aplicação móvel e sensível ao contexto que utiliza o compartilhamento de dados contextuais dos usuários do sistema, implementada utilizando o serviço do COP. Essa aplicação é baseada no cenário motivador apresentado na Seção 4.1. O MyPhotos é semelhante ao aplicativo Instagram[1], utilizado para aplicação de filtros em fotos e seu compartilhamento em redes sociais. Os usuários podem compartilhar fotos e marcá-las com *hashtags*, de modo a facilitar a busca por outros usuários. Além disso, fotos também são marcadas com a localização e o instante (data/hora) em que foram capturadas pelo dispositivo móvel do usuário. Uma vez que o serviço guarda os dados de vários usuários, a aplicação MyPhotos sugere *hashtags* de fotos com contexto semelhante (mesmo local e instante). Alguns *screenshots* da aplicação são apresentados na Figura 22.

A Figura 22(a) mostra uma imagem capturada na aplicação e a opção de aplicação de dois filtros de imagem. A Figura 22(b) mostra o efeito do filtro de imagem chamado de *Complex*, que deixa a foto como aspecto de desenho (*cartoonizer*), e a Figura 22(c) mostra o efeito do filtro de imagem chamado de *Simplex*, que deixa a foto com tom vermelho. A Figura 22(d) mostra a configuração de localização e instante que serão consideradas como contexto semelhante para a recuperação das 5 *hashtags* mais comentadas nesse contexto e algumas *hashtags* que foram recuperadas.

5.1.1 Implementação do MyPhotos

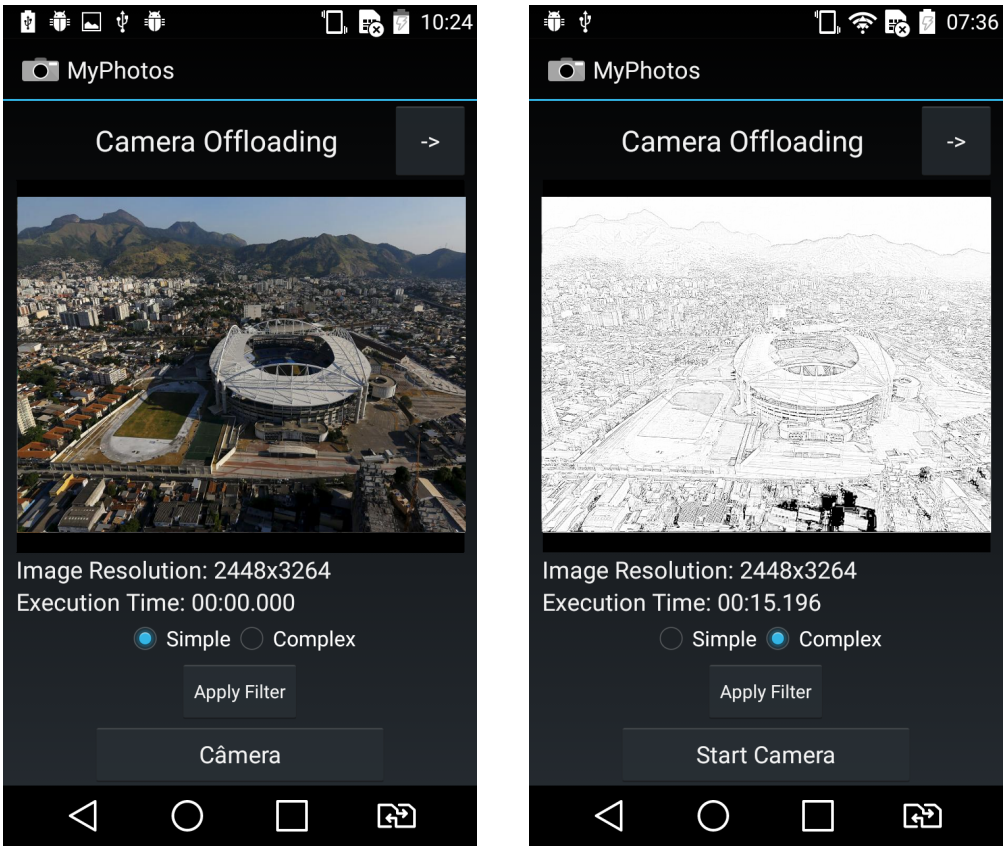
A Figura 23 apresenta um diagrama, que contém os 8 passos necessários para a implementação da prova de conceito.

O primeiro passo para o desenvolvimento do MyPhotos foi implementar os CACs que fornecem os dados contextuais necessários para a aplicação. Foram desenvolvidos dois CACs: o primeiro CAC fornece a localização do dispositivo móvel, enquanto o segundo encapsula um sensor lógico. Este segundo CAC utiliza os dados contextuais do primeiro CAC e fornece dois valores: as *hashtags* produzidas pelo usuário e a localização/instante em que elas foram produzidas.

Para a configuração do CAC é necessário configurar alguns parâmetros que serão

[1] https://instagram.com

Figura 22 – *Screenshots* da Aplicação MyPhotos

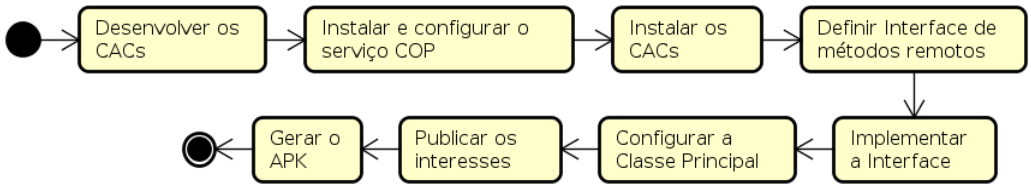


(a) (b)

(c) (d)

Fonte: Elaborado pelo autor

Figura 23 – Diagrama de Implementação do MyPhotos



Fonte: Elaborado pelo autor

utilizados pelo serviço proposto. O principal parâmetro do CAC é a *ContextKey* (CK), que informa o tipo de dado contextual que ele provê. Outro parâmetro que é utilizado no COP é a frequência de publicação dos dados contextuais pelo CAC. Para o primeiro CAC, sua CK foi definido como "context.device.location". Esse CAC provê dados contextuais contendo a latitude e a longitude da localização do dispositivo móvel. Para o segundo CAC, a CK foi definida como "context.ambient.hashtags".

O CAC de *hashtags* foi projetado para que sempre que o usuário da aplicação do MyPhotos entrar com as *hashtags*, produzir dados contextuais contendo a localização do dispositivo móvel e as *hashtags*. Assim, o CAC de *hashtags* realiza uma subscrição das *hashtags* produzidas e uma leitura da localização do dispositivo e fornece um novo dado contextual com ambas as informações.

Depois do desenvolvimento dos CACs necessários, o segundo passo foi instalar e configurar o serviço do COP no dispositivo móvel (ver Seção A). Depois de instalado o serviço, os CACs desenvolvidos para a aplicação do MyPhotos foram instalados no dispositivo móvel no terceiro passo. Para isso, cada CAC foi colocado numa pasta específica do dispositivo móvel criada depois da instalação do serviço do COP.

O quarto passo foi definir a interface do método candidato à realização de *offloading*. Este método executa o filtro contextual no ambiente remoto (ver Seção A). Nesse método, o filtro contextual recupera as *hashtags* que possuem contexto semelhante ao do usuário (i.e., localização/instante definidos). A Listagem 2 apresenta o trecho de código dessa interface.

Listagem 2 – Trecho de Código-Fonte da Interface do MyPhotos

```
1  public interface ITags {
2    @Remotable
3    public String readTags(double latitude, double longitude, double intervalTime, double
         intervalLocation);
4  }
```

O quinto passo foi criar uma classe que implemente essa interface e o método que irá recuperar os dados contextuais. A Listagem 3 apresenta trechos do método implementado, que consiste no filtro local e remoto para recuperação das *hashtags* com contexto semelhante ao do usuário.

Listagem 3 – Trecho de Código-Fonte da Classe que Implementa a Interface do MyPhotos

```
1  public class Tags implements ITags {
2    ...
3    @Override
4    public String readTags(double latitude, double longitude, double intervalTime, double
         intervalLocation) {
5      ...
6      Pattern p = (Pattern) new Pattern();
7      p.addField("ContextKey", "context.ambient.hashtags");
8      result = (ArrayList) COPManager.read(p, f);
9      ...
10   }
11   ...
12   IFilter.Stub filter = new IFilter.Stub() {
13     @Override
14     public boolean localFilter(Tuple tuple) throws RemoteException {
15       return isSimilar(tuple);
16     }
17
18     @Override
19     public boolean remoteFilter(Tuple tuple) throws RemoteException {
20       return isSimilar(tuple);
21     }
22   };
23   ...
24 }
```

O sexto passo foi configurar a classe principal do aplicativo, que é a primeira *activity*, com a integração do processo de inicialização do COP e MpOS na aplicação móvel e implementar

a chamada ao método remoto. A Listagem 4 apresenta o trecho de código da classe principal.

Listagem 4 – Trecho de Código-Fonte da Classe Principal do MyPhotos

```
1  @MposConfig
2  public class MainActivity extends Activity {
3    @Inject(Tags.class)
4    ITags tags;
5
6    COPManager cop;
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     ...
12     Mpos.getInstance().start(this, this);
13     cop = new COPManager(this, getPackageName());
14     cop.start(this);
15     ...
16     tags.readTags(latitude, longitude, timeField, locationField);
17     ...
18   }
19   ...
```

Uma vez que o COP esteja em funcionamento, com CACs instalados e a classe principal configurada, o sétimo passo foi publicar os interesses da aplicação no SysSU, que iniciará a aquisição de contexto com os CACS desenvolvidos, além de realizar consultas por dados contextuais. A aplicação MyPhotos tem interesse na localização do dispositivo móvel e nas *hashtags*. A Listagem 5 apresenta o trecho de código que publica o interesse nesses dados contextuais no SysSU. O único parâmetro necessário é a CK do CAC responsável por fornecer os dados contextuais de interesse.

Listagem 5 – Trecho de Código-Fonte de Publicação do Interesse do MyPhotos

```
1  cop.putInterest("context.device.location");
2  cop.putInterest("context.ambient.hashtags");
```

Por fim, o último passo é gerar o *Android Application Package* (APK) da aplicação e depois colocá-lo na pasta onde os APKs são armazenados no servidor, para que o *framework* MpOS seja capaz de realizar as operações de *offloading* (ver Seção A).

5.1.2 Descrição do Experimento

O experimento para testes do MyPhotos tem por objetivo demonstrar o funcionamento do *offloading* de dados contextuais e o mecanismo de privacidade para esses dados no serviço proposto. Foram utilizados dois *smartphones* e um *laptop*. O *laptop*, que funciona como *cloudlet*, tem as seguintes configurações: ASUS X450LD com processador Intel Core i5-4200U 1.60 GHz Quad Core, 8 GB de memória RAM, 1 TB de disco rígido e sistema operacional Linux Mint 17.2 Rafaela. O primeiro *smartphone* foi um LG G3 Beat com chipset Qualcomm Snapdragon 400 MSM8226 Cortex-A7 1.2 GHz Quad Core, memória interna 8GB e 1 GB de memória RAM, executando Android 5.0.2. O segundo *smartphone* foi um Motorola Moto G1 com chipset Qualcomm Snapdragon 400 MSM8226 1.2 GHz Quad Core, memória interna 16GB e 1 GB de memória RAM, executando Android 5.1.

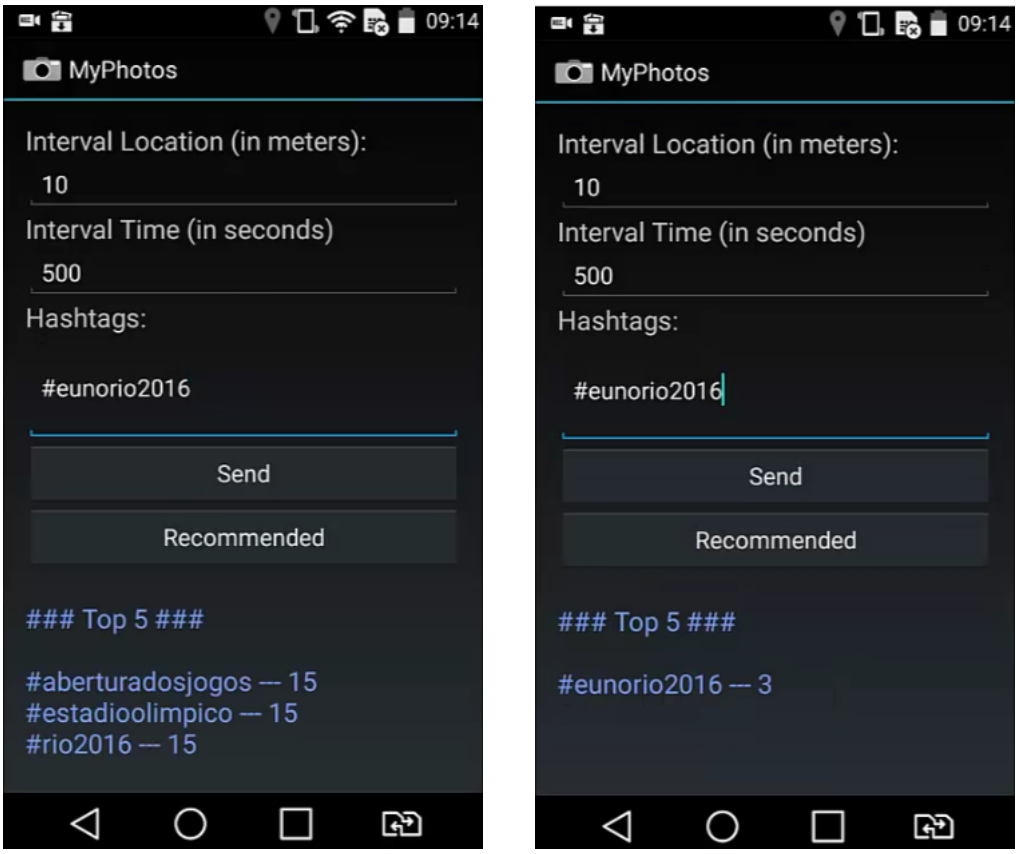
Foram ajustadas as configurações necessárias ao serviço do COP em cada *smartphone*. Como dito anteriormente, foi necessário configurar alguns parâmetros para a recomendação das *hashtags*, o intervalo de localização (em metros) e o intervalo de tempo (em segundos). Assim, o usuário define o quão distante e o quão atrasado estão os dados referentes à sua posição geográfica atual. O *cloudlet* que recebe as requisições dos *smartphones* foi configurado como "não confiável" para os dois dispositivos móveis.

5.1.3 Resultados Obtidos

A partir das configurações tanto do serviço COP como da aplicação MyPhotos, foram realizadas as interações de publicação e recuperação das *hashtags*. Um vídeo exemplificando as interações na aplicação e no serviço está disponível no seguinte endereço: <https://youtu.be/-bQSmBSCFuU>. A Figura 24 mostra os *screenshots* da aplicação MyPhotos no primeiro dispositivo.

A Figura 22(a) foi utilizada nos dois dispositivos para aplicação do filtro de imagem. O primeiro dispositivo (D1) foi configurado para não tornar público seus dados contextuais. Em D1 foi aplicado o filtro *Complex*. A Figura 24(a) mostra que D1 está conectado. Nessa tela foram recomendadas algumas *hashtags* (#estadioolimpico, #aberturadosjogos e #rio2016). Nesta execução foi inserida uma nova *hashtag*: #eunorio2016. Como D1 foi configurado para não tornar público os dados, eles não foram enviados para o *cloudlet*, como pode ser visto na

Figura 24 – MyPhotos em Execução no Primeiro Dispositivo Móvel



(a) Tela de recomendação com conexão (b) Tela de recomendação sem conexão

Fonte: Elaborado pelo autor

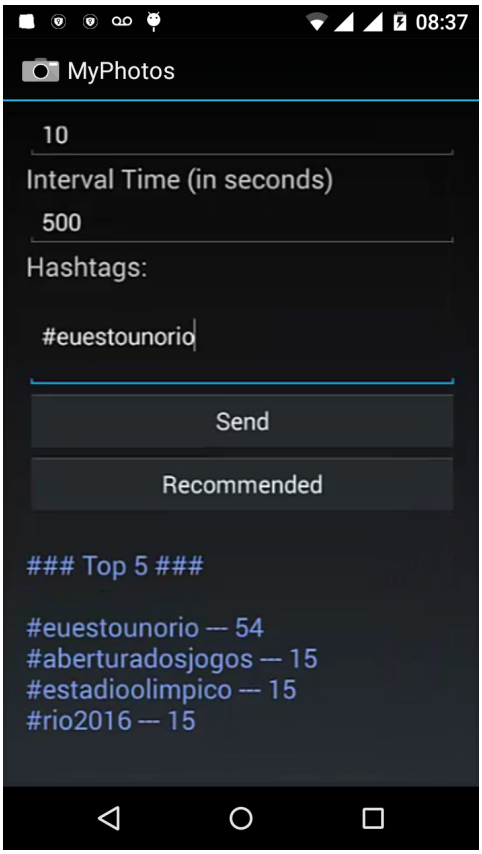
Figura 24(a), o qual possui todas as *hashtags* recomendadas menos a *hashtag* inserida.

No COP, caso o dispositivo não esteja conectado ao ambiente remoto, os dados contextuais recuperados pelos filtros contextuais são apenas os contidos no próprio dispositivo móvel. Como as *hashtags* de D1 não foram enviadas, a recomendação da aplicação são as próprias inseridas por ele (#eunorio2016) (ver Figura 24(b)).

O segundo dispositivo (D2) foi configurado para tornar público seus dados contextuais. Em D2 foi aplicado o filtro *Simplex*, e enviada a *hashtag* #euestounorio. Quando solicitada a recomendação de *hashtags*, foram recomendadas as mesmas que tinham sido recomendadas para D1, quando ele estava conectado, acrescidas de #euestounorio, como ilustrado na Figura 25.

Durante as requisições de recomendação de *hashtags*, o método *readTags* apresentado na Listagem 3 é invocado. No servidor foi implementado um esquema de monitoramento das requisições de *offloading*. A Figura 26 mostra uma visão da requisição do método *readTags*

Figura 25 – MyPhotos em Execução no Segundo Dispositivo Móvel



Fonte: Elaborado pelo autor

no servidor. É possível ver que foi realizado o *offloading* no método invocado da classe no qual ele foi implementado.

Figura 26 – Requisição ao Método Remoto no Servidor

```
##################
### OFFLOADING ###
##################
Making offloading for:
==================================
method = readTags
className = br.ufc.great.myphotos.Tags
==================================

2016/12/16 13:28:39.923 [ INFO] Thread-5 (AuthenticationService.java) - Device detected: motorola XT1033 (192.168.0.104)
```

Fonte: Elaborado pelo autor

Portanto, o usuário consegue recomendações de *hashtags* a partir de dados contextuais de outros usuários com o MyPhotos. Essa aplicação mostra a importância da migração dos dados para um ambiente centralizador, ao mesmo tempo que torna transparente para o desenvolvedor e configurável por parte do usuário, validando a Hipótese 1.

5.2 Avaliação do Impacto do Processamento dos Filtros Contextuais

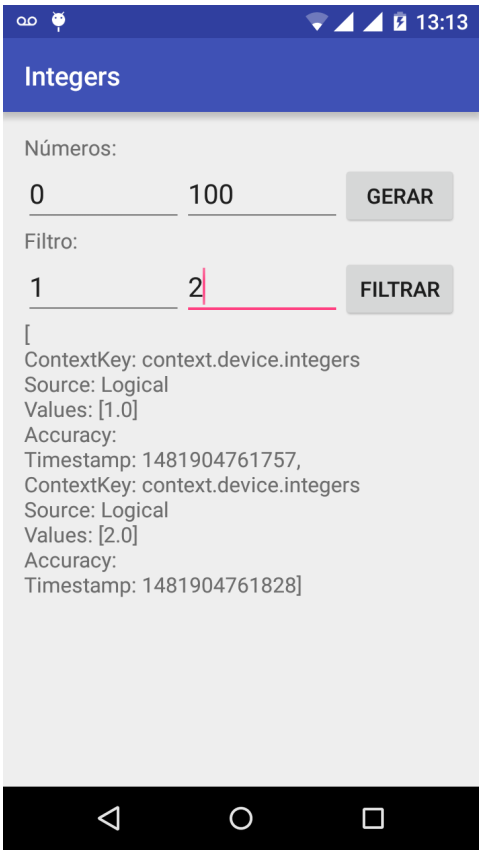
Além do desenvolvimento da aplicação MyPhotos, também foi idealizada uma aplicação móvel e sensível ao contexto chamada *Integers*, para avaliação do impacto do processamento dos filtros contextuais para a recuperação dos dados contextuais tanto no dispositivo móvel como em um *cloudlet*. Essa aplicação foi utilizada para realização de testes que avaliam o tempo de processamento e o gasto energético desses filtros no dispositivo móvel e em um *cloudlet*.

A aplicação *Integers* foi idealizada para prover dados contextuais contendo valores de inteiros. Ela não tem funcionalidade prática, sendo implementada apenas para efeito de testes. Para essa aplicação foi desenvolvido um CAC chamado de *IntegersCAC*, que recebe dois limiares, um inferior e outro superior, e gera tuplas de dados contextuais para esse intervalo. Por exemplo, se o CAC receber o valor 0 (zero) como limiar inferior e o valor 100 (cem) como limiar superior, ele irá gerar tuplas a partir de um laço contendo esse intervalo e incrementando o valor até chegar ao limiar superior (i.e., 0, 1, 2, ..., 100). Assim, no final da execução desse laço, o COP, mais especificamente o SysSU, terá 101 tuplas de dados contextuais contendo valores inteiros.

A Figura 27 apresenta uma tela capturada da aplicação *Integers*. Nessa aplicação, o usuário entra com o limiar inferior e superior do intervalo com o qual ele quer gerar as tuplas e pressiona o botão chamado GERAR. Para executar o filtro contextual em cima dessas tuplas, o usuário também deve entrar com o limiar inferior e superior do intervalo o qual ele quer recuperar os dados contextuais e pressionar o botão chamado FILTRAR. Assim, todas as tuplas que estiverem nesse intervalo deverão ser recuperadas do dispositivo móvel ou do ambiente remoto.

A aplicação *Integers* é simples, tanto em termos de concepção como de implementação, mas ela é útil para o experimento de processamento dos filtros contextuais. Como o usuário consegue controlar a quantidade de tuplas geradas no COP, bem como controlar a quantidade de tuplas que serão recuperadas a partir dos filtros contextuais, então é possível saber exatamente a quantidade de dados recuperada do serviço proposto. A partir da execução dos filtros contextuais na aplicação, foi medido o tempo de processamento e o gasto energético.

Figura 27 – *Screenshot* da Aplicação *Integers*



Fonte: Elaborado pelo autor

5.2.1 Descrição do Experimento

O experimento foi realizado em duas etapas. A primeira etapa consistiu na medição do tempo e gasto energético de processamento dos filtros contextuais no dispositivo móvel, enquanto que na segunda etapa foi realizado a mesma medição em um *cloudlet*.

Para calcular o tempo de processamento de execução dos filtros contextuais foi medido o tempo gasto a partir da chamada ao método responsável pela recuperação das tuplas, até o retorno da execução desse método. Esse tempo foi calculado em milissegundos (ms). Para calcular o gasto energético da execução desses filtros foi utilizado um aparelho chamado de *Power Monitor*[2], um monitor que realiza a análise do consumo energético do dispositivo móvel, e eliminado o máximo possível de recursos do dispositivo que pudessem prejudicar o experimento. Uma das métricas analisadas no *Power Monitor* é a potência em Watts (W). O gasto energético foi calculado com a multiplicação da potência e do tempo de processamento, que equivale a energia em milijoules (mJ).

[2] https://www.msoon.com/LabEquipment/PowerMonitor/

Na primeira etapa foi utilizado um *smartphone* modelo LG G3 Beat, que também foi usado no experimento da prova de conceito descrito na Seção 5.1.2. Foi requisitada a criação de 10000 tuplas, no intervalo de 1 até 10000. Para a execução dos filtros contextuais, foi realizada a filtragem em passos de 100, do limiar inferior 0 até o limiar superior de 5000. Assim, as filtragens foram da seguinte forma: 1 a 100, 1 a 200, 1 a 300, ..., 1 a 5000. Para cada filtragem, foram medidos o tempo de processamento e o gasto energético.

Na segunda etapa foram utilizados três *smartphones* e um *laptop*. O primeiro e o segundo *smartphones* foram os mesmos utilizados no experimento da prova de conceito. Por fim, o terceiro *smartphone* foi um Lenovo Vibe K5 com chipset Qualcomm Snapdragon 616 MSM8939v2 1.4 GHz Octa Core, memória interna 16GB e 2 GB de memória RAM, executando Android 5.1. O *laptop*, que funciona como *cloudlet*, foi o mesmo do experimento da prova de conceito. No primeiro *smartphone*, o intervalo para geração de tuplas foi definido de 1 até 5000. No segundo *smartphone*, o intervalo para geração de tuplas foi definido de 5001 até 10000. Assim, foram geradas 10000 tuplas como na primeira etapa. Para a execução dos filtros contextuais, foi utilizado o terceiro *smartphone* e realizado a filtragem em passos de 100, do limiar inferior 0 até o limiar superior de 5000. Assim, as filtragens foram da seguinte forma: 1 a 100, 1 a 200, 1 a 300, ..., 1 a 5000. Para cada filtragem, foram medidos o tempo de processamento e o gasto energético.

De acordo com o Teorema Central do Limite (TCL) (FISCHER, 2010), quando se tem uma amostra suficientemente grande, a distribuição de probabilidade da média amostral pode ser aproximada por uma distribuição normal. Essa aproximação é valida a partir de 30 médias amostrais. Assim para cada uma das etapas, os experimentos foram realizados 30 vezes e os dados coletados foram armazenados em uma planilha.

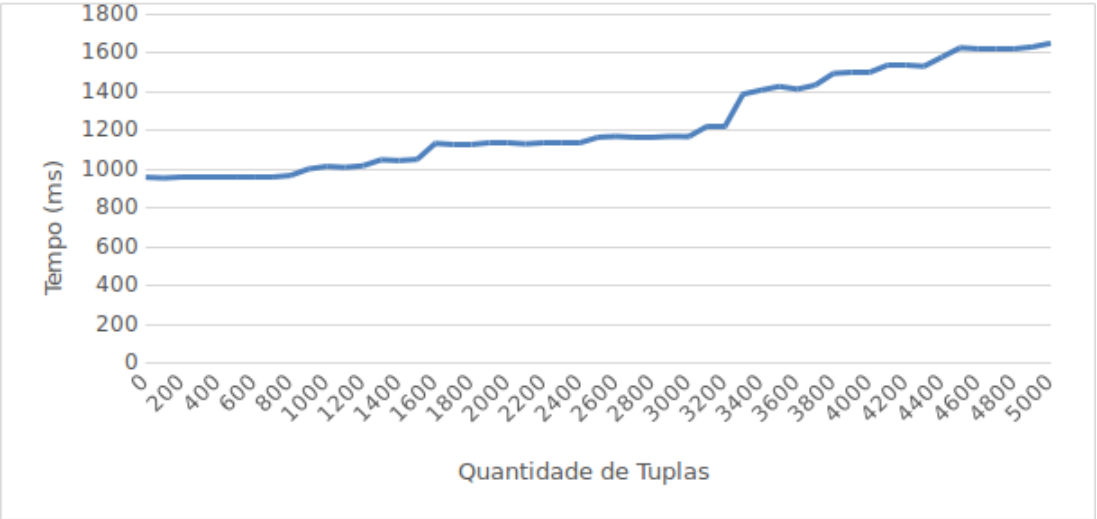
5.2.2 Resultados Obtidos

Uma vez que os dados foram coletados e armazenados em uma planilha, foi realizada uma análise dos dados brutos e discutido sobre as inferências obtidas através desses dados. Tanto para o tempo de processamento quanto para o gasto energético dos filtros contextuais de cada etapa, foi feita a média aritmética das 30 vezes que os experimentos foram executados.

A Figura 28 apresenta um gráfico com o resultado do processamento do filtro

contextual no *cloudlet*. O gráfico mostra o tempo de processamento em milissegundos pela quantidade de tuplas recuperadas a partir das tuplas geradas. A partir do gráfico, é possível verificar que o tempo de processamento tem pouca variação com o aumento da quantidade de tuplas recuperadas. Assim, pode-se afirmar que a qualidade da conexão tem maior influência no serviço do que a quantidade de tuplas processadas.

Figura 28 – Tempo de Processamento do Filtro Contextual no *Cloudlet*



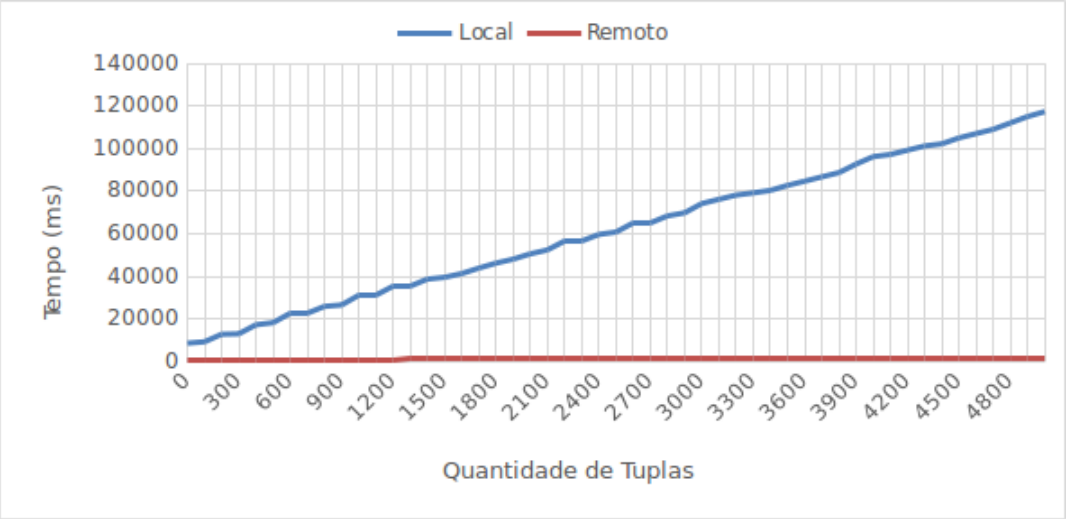
Fonte: Elaborado pelo autor

A Figura 29 apresenta um gráfico comparativo do tempo de processamento do filtro contextual no dispositivo móvel (local) e no *cloudlet* (remoto). A partir do gráfico é possível verificar que o tempo de processamento no dispositivo móvel é bastante superior se comparado ao do *cloudlet*.

Em relação ao gasto energético do processamento dos filtros contextuais, a Figura 30 apresenta a relação entre a energia gasta em milijoules (mJ) pela quantidade de tuplas recuperadas a partir das tuplas geradas no *cloudlet*. Como a energia depende do tempo, esse gráfico tem o mesmo comportamento do gráfico da Figura 28. Assim, é possível verificar que o gasto energético mostrou um crescimento suave com o aumento da quantidade de tuplas, apresentando apenas pequenas oscilações.

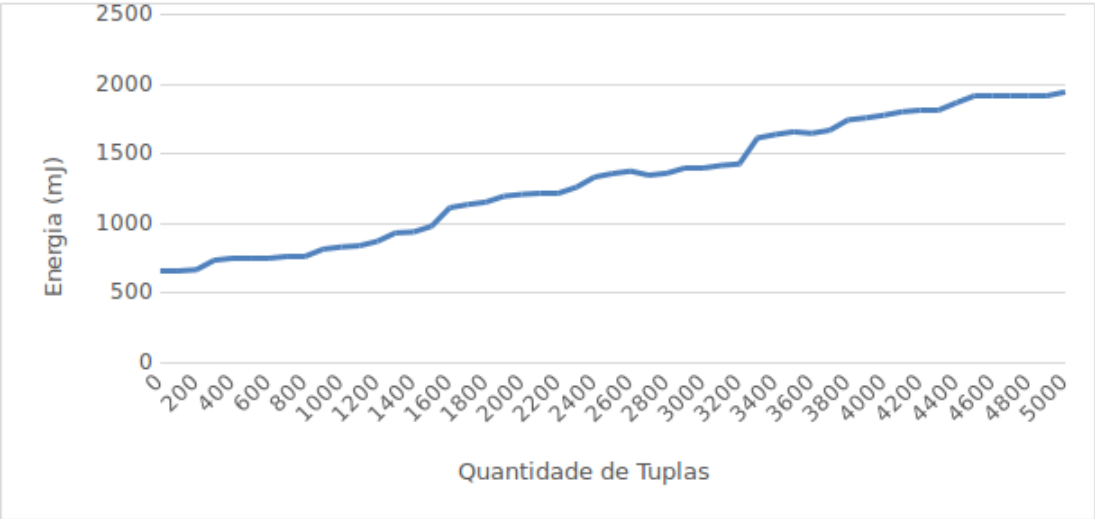
A Figura 31 apresenta um gráfico comparativo do gasto energético do processamento do filtro contextual no dispositivo móvel (local) e no *cloudlet* (remoto). A partir do gráfico, é possível verificar que o gasto energético de processamento no dispositivo móvel é bastante superior se comparado ao do *cloudlet*.

Figura 29 – Tempo de Processamento do Filtro Contextual no Dispositivo Móvel e no *Cloudlet*



Fonte: Elaborado pelo autor

Figura 30 – Gasto Energético do Processamento do Filtro Contextual no *Cloudlet*



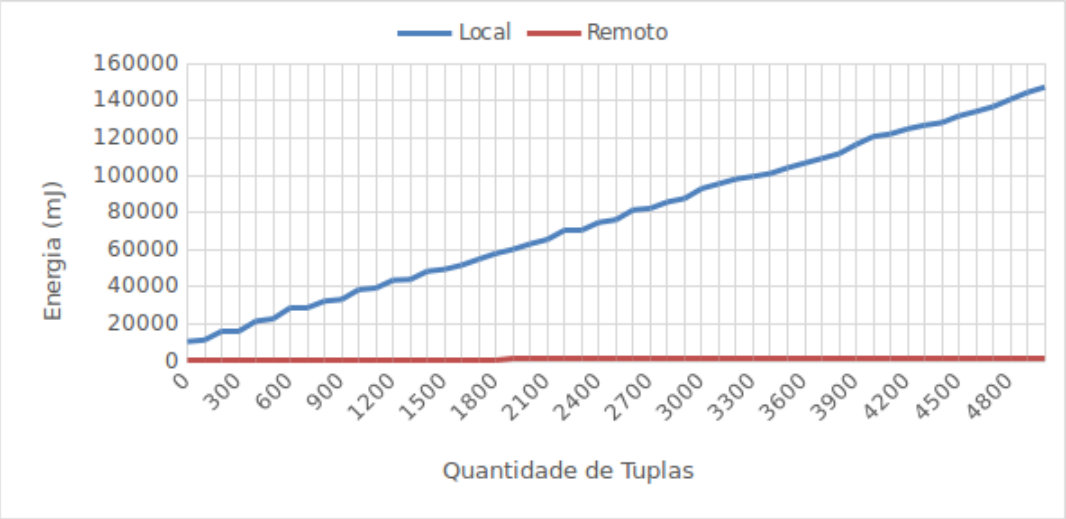
Fonte: Elaborado pelo autor

A partir dos gráficos apresentados nesta seção, pode-se confirmar que a migração de dados do dispositivo móvel para um ambiente remoto é vantajosa tanto em termos de retirar o armazenamento desse dispositivo como ter maior velocidade e menos gasto energético na recuperação desses dados, validando a Hipótese 2.

5.3 Avaliação do Impacto das Políticas de Sincronização

O último experimento realizado foi uma avaliação do impacto das políticas de sincronização no envio do dados contextuais do dispositivo móvel para um ambiente remoto.

Figura 31 – Gasto Energético do Processamento do Filtro Contextual no Dispositivo Móvel e no *Cloudlet*



Fonte: Elaborado pelo autor

Para isso, um novo aplicativo foi desenvolvido, chamado *Sending*. Esse aplicativo foi utilizado para realização de testes que avaliam o gasto energético das estratégias de envio desses dados para um *cloudlet*.

A aplicação *Sending* foi idealizada para prover dados contextuais do acelerômetro. Ela não tem funcionalidade prática, sendo implementada apenas para efeito de testes. Para essa aplicação foram desenvolvidos três CACs que fornecem valores do acelerômetro do dispositivo móvel em três frequências de publicação diferentes. Para definir a frequência com que os dados contextuais eram publicados por tais CACs, foi utilizado a documentação do Android sobre sensores[3].

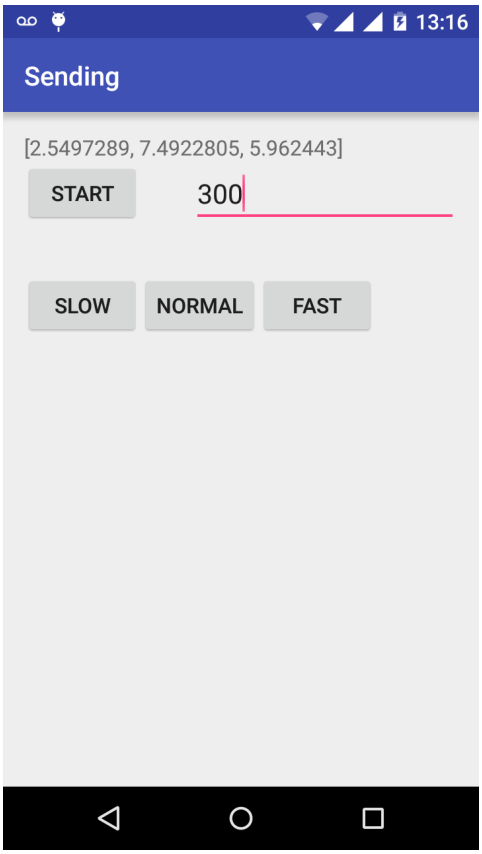
Nessa documentação, são definidas constantes que correspondem ao tempo que os sensores do dispositivo devem levar para publicar as informações. A constante "SENSOR_DELAY_NORMAL" define o tempo de 200 ms. Assim, se essa constante for utilizada, o sensor deve publicar as informações aproximadamente nesse tempo. Levando em consideração esse valor da constante, foram definidos três modos de frequência de publicação dos CACs:

- **Lento**: são todas as frequências de publicação maior que 200 ms;
- **Normal**: são todas as frequências de publicação igual a 200 ms;
- **Rápido**: são todas as frequências de publicação menor que 200 ms

[3] https://developer.android.com/guide/topics/sensors/sensors_overview.html

Seguindo estes modos, foi desenvolvido um CAC de acelerômetro para cada modo. Na aplicação, o usuário entra com um valor que define o período em segundos em que cada estratégia de envio definida no serviço do COP será executada e o CAC que irá fornecer as tuplas de acelerômetro. A Figura 32 mostra uma tela capturada da aplicação *Sending* em execução. Depois de definir a estratégia utilizada, o tempo de monitoramento e o CAC, ele deve pressionar o botão START. Os números acima do botão START são valores fornecidos pelo CAC de acelerômetro.

Figura 32 – *Screenshot* da Aplicação *Sending*



Fonte: Elaborado pelo autor

5.3.1 Descrição do Experimento

O experimento foi realizado para cada uma das três estratégias de envio dos dados contextuais. Para cada estratégia, foram utilizados os três CACs de acelerômetro diferentes (i.e., lento, normal e rápido). O experimento consistiu na medição do gasto energético do envio dos dados contextuais do dispositivo móvel para um *cloudlet*.

No experimento, foram utilizados o *smartphone* LG G3 Beat e o *laptop* do experi-

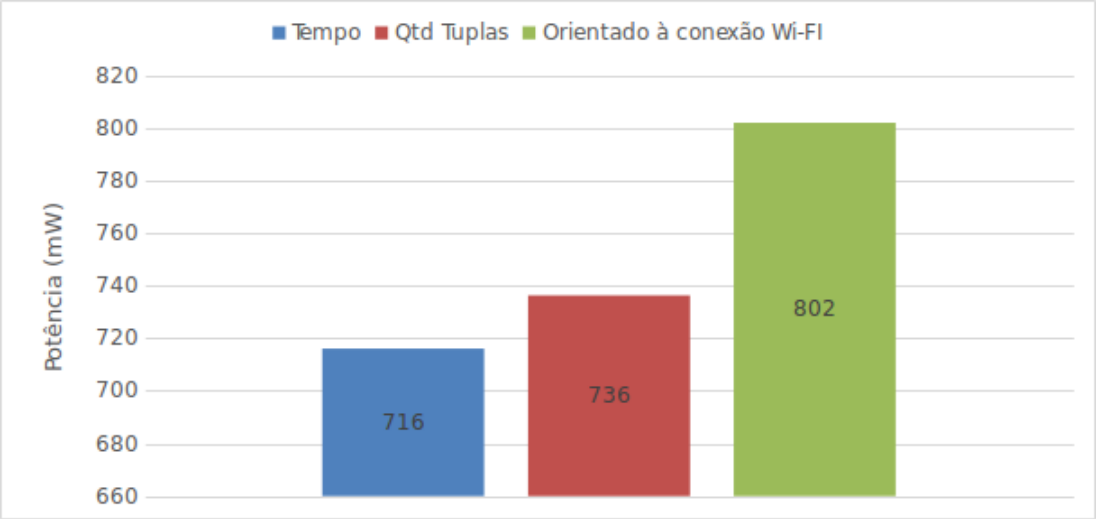
mento da prova de conceito. Para cada estratégia de envio foi utilizado o *Power Monitor* para medir o gasto energético durante 300 segundos.

5.3.2 Resultados Obtidos

Assim como no experimento descrito na Seção 5.2, os dados foram coletados e armazenados em uma planilha, para análise dos dados brutos e realização de inferências por meio de tais dados. Para cada estratégia de envio e CAC de acelerômetro, foi medido o gasto energético 30 vezes e feito a média aritmética.

A Figura 33 apresenta um gráfico com o resultado do gasto energético do dispositivo móvel utilizando o CAC de acelerômetro no modo lento. O gráfico mostra a potência média em miliwatts (mW) para cada estratégia de envio de dados contextuais em 300 segundos. A partir do gráfico, é possível verificar que a estratégia de envio por tempo é a que possui um menor gasto energético, enquanto o envio por orientação à conexão *Wi-Fi* é a que consome mais energia.

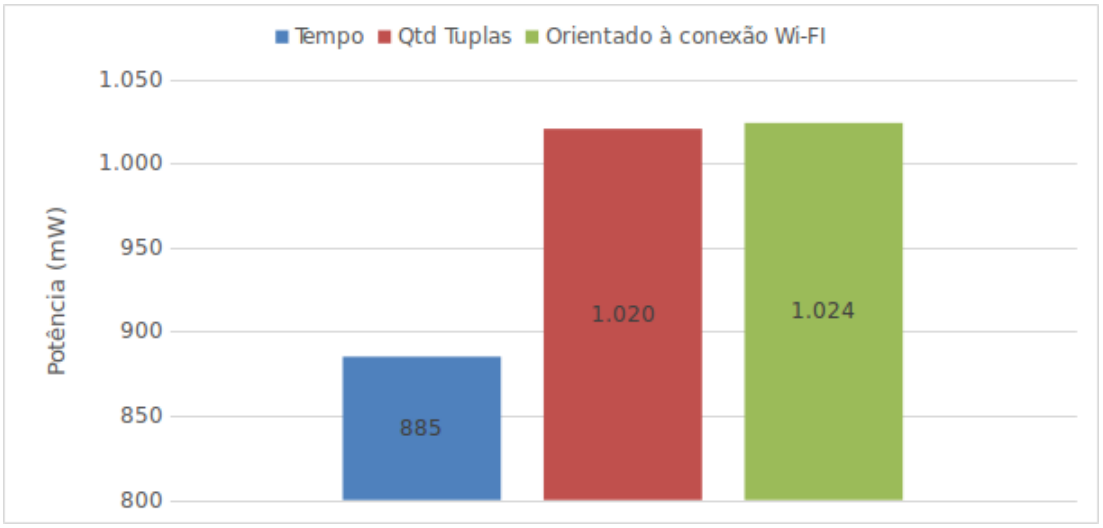
Figura 33 – Gasto Energético do Dispositivo Móvel Utilizando o CAC de Acelerômetro no Modo Lento



Fonte: Elaborado pelo autor

A Figura 34 apresenta um gráfico com o resultado do gasto energético do dispositivo móvel utilizando o CAC de acelerômetro no modo normal. Esse gráfico mostra o mesmo comportamento em relação ao gráfico anterior, em que a estratégia de envio por tempo é a que possui menor gasto energético e a estratégia de envio por orientação à conexão *Wi-Fi* é a que consume mais energia.

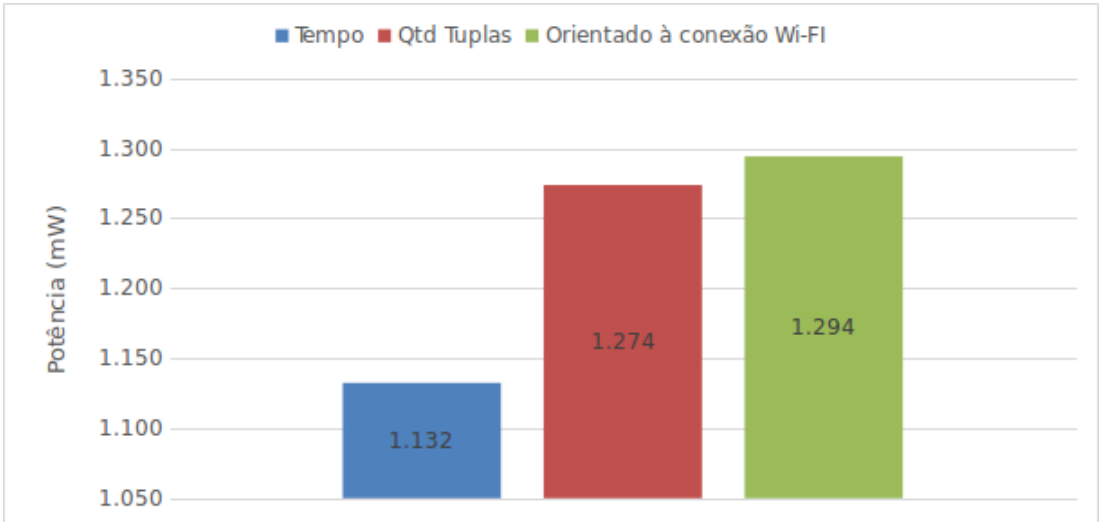
Figura 34 – Gráfico de Gasto Energético do Dispositivo Móvel Utilizando o CAC de Acelerômetro no Modo Normal



Fonte: Elaborado pelo autor

O mesmo comportamento acontece também quando é utilizado o CAC de acelerômetro no modo rápido, como mostra a Figura 35. Porém observa-se que no geral, todas as estratégias tem um aumento no consumo de energia em relação ao gráfico anterior.

Figura 35 – Gráfico de Gasto Energético do Dispositivo Móvel Utilizando o CAC de Acelerômetro no Modo Rápido



Fonte: Elaborado pelo autor

A partir dos gráficos apresentados nesta seção, pode-se confirmar que a estratégia de envio por tempo é a que consome menos energia e a estratégia de envio por orientação à conexão *Wi-Fi* é a que consome mais energia. Esse comportamento é justificável, pois na estratégia de

envio por tempo, o COP só tenta enviar os dados para o servidor periodicamente, diminuindo a quantidade de requisições.

A estratégia de envio por quantidade de tuplas, nesse experimento, realiza muitas requisições ao servidor, assim o consumo energético é tão ruim quanto na estratégia de envio por conexão *Wi-Fi*, que fica constantemente monitorando se o *Wi-Fi* foi habilitado no dispositivo e precisa enviar uma quantidade de dados maior do que as outras estratégias, pois os dados só são enviados quando a conexão for estabelecida.

5.4 Comparativo com os Trabalhos Relacionados

A Tabela 2 posiciona o COP proposto em relação aos trabalhos relacionados apresentados no Capítulo 3. O COP foi projetado de maneira a conseguir satisfazer a lacuna encontrada nesses trabalhos, ou seja uma solução que consiga prover um mecanismo configurável de envio e de privacidade dos dados contextuais do dispositivo móvel para o ambiente remoto.

Tabela 2 – Comparativo Entre os Trabalhos Relacionados e o COP

Trabalho	Tipo	Plataforma	Paradigma de Comunicação	Modelo de Contexto	Política de Sincronização	Privacidade
CUPUS	*Middleware*	Android	*Publish-Subscribe*	Chave-Valor	Sim	Não
Sensarena	*Framework*	Android	*Request-Response*	Chave-Valor	Não	Sim
CAROMM	*Framework*	Android	*Request-Response*	Linguagem de Marcação	Sim	Não
Sahyog	*Middleware*	Android	*Request-Response* *Publish-Subscribe*	Chave-Valor	Não	Sim
COP	*Middleware*	Android	*Request-Response* *Publish-Subscribe*	Chave-Valor	Sim	Sim

Fonte: Elaborado pelo autor

5.5 Considerações Finais

Este capítulo apresentou uma aplicação móvel e sensível ao contexto que utiliza o compartilhamento de dados contextuais desenvolvida sobre o COP como uma prova de conceito, que é baseada no cenário motivador apresentado na Seção 4.1, e testes realizados no mecanismo de privacidade do serviço proposto. Também foram realizados testes em relação ao impacto do processamento dos filtros contextuais sobre o local de execução e em relação ao impacto das políticas de sincronização no gasto energético do dispositivo móvel. Por fim, foi realizado um comparativo entre os trabalhos relacionados e o COP.

Na prova de conceito foi demonstrado passo a passo como desenvolver uma aplicação móvel e sensível ao contexto com compartilhamento de dados utilizando o COP. Com a prova de conceito foi possível perceber como é importante a migração dos dados para um ambiente centralizador, assim como ter um serviço transparente para o desenvolvedor e um mecanismo de privacidade configurável por parte do usuário.

Além da prova de conceito, também foi implementada uma aplicação chamada *Integers*, para realização de testes no impacto do processamento dos filtros contextuais. O experimento consistia em medir o tempo e o gasto energético do processamento desses filtros no dispositivo móvel e no *cloudlet*. O experimento detectou que a migração de dados do dispositivo móvel para um ambiente remoto é vantajosa tanto em termos de retirar o armazenamento desse dispositivo como ter maior velocidade e menos gasto energético na recuperação desses dados.

Para avaliar as políticas de sincronização, foi realizado um teste de impacto do gasto energético das diferentes estratégias implementadas para o serviço proposto. O experimento detectou que a estratégia de envio por tempo é a que consome menos energia e a estratégia de envio por orientação à conexão *Wi-Fi* é a que consome mais energia.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo resume o que foi discutido e desenvolvido nesta dissertação de mestrado. A Seção 6.1 apresenta os resultados alcançados com a realização da proposta, enquanto na Seção 6.2 é apresentada a principal limitação encontrada na solução. A Seção 6.3 apresenta as publicações durante o período do mestrado e a Seção 6.4 discorre sobre os trabalhos futuros para melhorias deste trabalho.

6.1 Resultados Alcançados

Este trabalho apresentou um serviço de *offloading* de dados contextuais com suporte a privacidade denominado de COP. Esse serviço amplia as possibilidades para os desenvolvedores de aplicações móveis e sensíveis ao contexto que utilizam o compartilhamento de dados contextuais dos usuários do sistema, bem como prover uma mecanismo de privacidade para que o usuário possa definir se os dados dele devem ser compartilhados com os outros usuários.

O trabalho proposto baseia-se em um modelo de contexto, uma política de privacidade e em políticas de sincronização. O modelo de contexto foi estendido do LoCCAM, o qual foi alterado para que a solução pudesse lidar com a migração de dados contextuais. Na solução, os dados migrados de todos os dispositivos do sistema são enviados para um ambiente centralizado de armazenamento e processamento em nuvem.

Em relação à política de privacidade da solução proposta, foram estabelecidos dois conceitos: dados contextuais sensíveis e *cloudlet* confiável. Os dados contextuais sensíveis estão relacionados com a visibilidade desses dados (i.e., público e privado). O *cloudlet* confiável ou *gateway* é definido pelo usuário (e.g., um *desktop* da casa desse usuário). Se os dados contextuais forem definidos como privados, eles são difundidos para um *cloudlet* confiável. Caso não seja um *gateway*, esses dados não são migrados.

Em relação ao momento em que os dados contextuais são enviados do dispositivo móvel para a nuvem, foram estabelecidas estratégias diferentes para a migração dos dados, os quais foram denominadas de políticas de sincronização. Atualmente na solução, três estratégias de envio encontram-se implementadas: por tempo, por quantidade de tuplas e orientado à conexão *Wi-Fi*.

Como prova de conceito deste trabalho de dissertação, foi apresentada a aplicação MyPhotos, construída sobre o COP, e como ela foi desenvolvida. Na implementação da prova de conceito, foram seguidos alguns passos necessários para toda a construção e execução da aplicação. O primeiro passo foi o desenvolvimento dos CACs que o MyPhotos utiliza. Depois, mostrou-se como o MyPhotos publica o interesses contextuais no COP, para que seja iniciada a aquisição e publicação dos dados contextuais. Depois, foi apresentada a interface com o método remoto utilizado na recuperação dos dados contextuais pelo MyPhotos, assim como a classe que implementa essa interface e os filtros contextuais tanto no dispositivo móvel como na nuvem. Por fim, foi apresentada a configuração utilizada na classe principal da aplicação. Depois de toda a implementação, foram realizados os testes de migração dos dados de dois dispositivos móveis para um *cloudlet*. Durante os testes, percebeu-se que é importante ter um serviço capaz de migrar os dados e que seja transparente para o desenvolvedor, assim como ter um mecanismo de privacidade configurável por parte do usuário.

Além da prova de conceito, foram desenvolvidas outras duas aplicações: *Integers* e *Sending*. A aplicação *Integers* foi implementada para avaliar o impacto do processamento dos filtros contextuais. O experimento consistia em medir o tempo e o gasto energético do processamento desses filtros no dispositivo móvel e no *cloudlet*. O experimento detectou que a migração de dados do dispositivo móvel para um ambiente remoto é vantajosa tanto em termos de retirar o armazenamento desse dispositivo como ter maior velocidade e menos gasto energético na recuperação desses dados.

A aplicação *Sending* foi desenvolvida para avaliar as políticas de sincronização implementadas. Nessa aplicação foi realizado um teste de impacto do gasto energético das três estratégias de envio. O experimento detectou que a estratégia de envio por tempo é a que consome menos energia e a estratégia de envio por orientação à conexão *Wi-Fi* é a que consome mais energia.

6.2 Limitação

Foi identificado uma limitação neste trabalho de dissertação de mestrado. Como o COP é uma extensão do *middleware* LoCCAM, e este se propõe a ser uma infraestrutura genérica, o modelo de contexto implementado também tenta ser genérico. Dessa forma, por apresentar um modelo mais simples e que não possibilita a representação de muitos relacionamentos

entre informações ou realização de inferências sofisticadas sobre ele, o nível semântico das informações diminui. Essa limitação pode ser contornada entregando para aplicação o uso de um modelo de contexto próprio.

6.3 Produção Bibliográfica

Durante o período do mestrado, foram realizados quatro publicações. Os dois últimos, apresentados a seguir, são relacionados com a proposta da dissertação:

- **CRITiCAL: *A Configuration Tool for Context Aware and mobiLe Applications***: Publicado como artigo completo no XXXIX Conferência Anual de *Software* e Aplicações de Computadores (COMPSAC), com Qualis A2, o trabalho apresenta uma abordagem para modelagem e geração de aplicações móveis e sensíveis ao contexto utilizando Engenharia baseada em Modelos e o LoCCAM. Autores: Paulo Artur de Sousa Duarte, Felipe Mota Barreto, Francisco Anderson De Almada Gomes, Windson Viana de Carvalho e Fernando Antonio Mota Trinta;
- ***Deployment* dinâmico para ambientes multimídia sensíveis ao contexto**. Publicado como artigo completo no XXI Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), com Qualis B3, o trabalho apresenta a evolução do LoCCAM, o qual executa dinamicamente a implantação de novos CACs, sendo capaz de da suporte a sistemas móveis e multimídias no cenário de Internet das Coisas. Este trabalho foi premiado como *best paper* de todo o evento. Autores: Paulo Artur de Sousa Duarte, Luís Fernando Maia Santos Silva, Francisco Anderson De Almada Gomes, Windson Viana de Carvalho e Fernando Antonio Mota Trinta;
- **Uma Proposta de Extensão de um *Middleware* de Aquisição de Contexto para Suporte a *Crowdsensing* com Privacidade**. Publicado no XII *Workshop* de Teses e Dissertações (WTD) do XXI Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), o trabalho apresenta um estágio intermediário da proposta do serviço COP. Autores: Francisco Anderson de Almada Gomes, Lincoln Souza Rocha e Fernando Antonio Mota Trinta; e
- **Um Serviço de *Offloading* de Dados Contextuais com Suporte a Privacidade**. Publicado como artigo completo no XXII Simpósio Brasileiro de Sistemas

Multimídia e Web (WebMedia), com Qualis B3, o trabalho apresenta o serviço COP proposto nesta dissertação. Este trabalho foi premiado como *best paper* na trilha de TV Digital, Computação Ubíqua e Móvel do evento. Autores: Francisco Anderson de Almada Gomes, Windson Viana de Carvalho, Lincoln Souza Rocha e Fernando Antonio Mota Trinta.

6.4 Trabalhos Futuros

Como trabalhos futuros, propõe-se algumas melhorias no COP, tais como:

- Executar o COP em uma nuvem pública, para que os dados contextuais presentes nos *cloudlets* sejam enviados para um local único, permitindo uma capacidade de armazenamento ainda maior, assim como de processamento dos filtros contextuais;
- Implementar um gerenciamento de escalabilidade do sistema, a fim de permitir que esse se adapte diante das requisições dos usuários;
- Realizar mais experimentos, levando em consideração mais métricas (e.g., memória interna), a fim de melhorar a avaliação do serviço proposto;
- Realizar uma avaliação do COP com desenvolvedores, a fim de analisar a qualidade da solução do ponto de vista dos engenheiros de *software* em termos de facilidade de uso, modularidade, dentre outros; e
- Criar um portal *web* contendo o serviço do COP, um conjunto de CACs para *download* e uma documentação extensa para uma maior visibilidade e participação de desenvolvedores.

REFERÊNCIAS

ANTONIć, A.; MARJANOVIć, M.; PRIPUžIć, K.; ŽARKO, I. P. A mobile crowd sensing ecosystem enabled by cupus: Cloud-based publish/subscribe middleware for the internet of things. **Future Generation Computer Systems**, v. 56, p. 607 – 622, 2016. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X15002575>.

BAJAJ, G.; SINGH, P. Sahyog: A middleware for mobile collaborative applications. In: **New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on**. [S.l.: s.n.], 2015. p. 1–5.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. **International Journal of Ad Hoc and Ubiquitous Computing**, Inderscience Publishers, v. 2, n. 4, p. 263–277, 2007.

CARROLL, A.; HEISER, G. An analysis of power consumption in a smartphone. In: **Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference**. Berkeley, CA, USA: USENIX Association, 2010. (USENIXATC'10), p. 21–21. Disponível em: <http://dl.acm.org/citation.cfm?id=1855840.1855861>.

COSTA, P. B.; REGO, P. A. L.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Mpos: A multiplatform offloading system. In: **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2015. (SAC '15), p. 577–584. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695945>.

DEY, A. K. Understanding and using context. **Personal Ubiquitous Comput.**, Springer-Verlag, London, UK, UK, v. 5, n. 1, p. 4–7, jan. 2001. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s007790170019>.

DUARTE, P. A.; SILVA, L. F. M.; GOMES, F. A.; VIANA, W.; TRINTA, F. M. Dynamic deployment for context-aware multimedia environments. In: **Proceedings of the 21st Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2015. (WebMedia '15), p. 197–204. ISBN 978-1-4503-3959-9. Disponível em: <http://doi.acm.org/10.1145/2820426.2820443>.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 1, p. 84–106, jan. 2013. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2012.05.023>.

FISCHER, H. **A history of the central limit theorem: From classical to modern probability theory**. [S.l.]: Springer Science & Business Media, 2010.

FONTELES, A. S. **Um framework para aquisição adaptativa e fracamente acoplada de informação contextual para dispositivos móveis**. Dissertação (Mestrado) — Universidade Federal do Ceará, 2013.

HENGARTNER, U.; STEENKISTE, P. Avoiding privacy violations caused by context-sensitive services. **Pervasive and mobile computing**, Elsevier, v. 2, n. 4, p. 427–452, 2006.

INDULSKA, J.; SUTTON, P. Location management in pervasive systems. In: AUSTRALIAN COMPUTER SOCIETY, INC. **Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21**. [S.l.], 2003. p. 143–151.

KHARBANDA, H.; KRISHNAN, M.; CAMPBELL, R. H. Synergy: A middleware for energy conservation in mobile devices. In: IEEE. **2012 IEEE International Conference on Cluster Computing**. [S.l.], 2012. p. 54–62.

KNAPPMEYER, M.; KIANI, S. L.; REETZ, E. S.; BAKER, N.; TONJES, R. Survey of context provisioning middleware. **IEEE Communications Surveys Tutorials**, v. 15, n. 3, p. 1492–1519, Third 2013. ISSN 1553-877X.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

LIMA, F.; ROCHA, L.; MAIA, P.; ANDRADE, R. A decoupled and interoperable architecture for coordination in ubiquitous systems. In: **Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on**. [S.l.: s.n.], 2011. p. 31–40.

LIU, J.; AHMED, E.; SHIRAZ, M.; GANI, A.; BUYYA, R.; QURESHI, A. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. **Journal of Network and Computer Applications**, Elsevier, v. 48, p. 99–117, 2015.

MAIA, M. E. F.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. Loccam - loosely coupled context acquisition middleware. In: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2013. (SAC '13), p. 534–541. ISBN 978-1-4503-1656-9. Disponível em: <http://doi.acm.org/10.1145/2480362.2480465>.

MESSAOUD, R. B.; REJIBA, Z.; GHAMRI-DOUDANE, Y. An energy-aware end-to-end crowdsensing platform: Sensarena. In: IEEE. **Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual**. [S.l.], 2016. p. 284–285.

NAVARRO, N. D. A. B.; COSTA, C. A. D.; BARBOSA, J. L. V.; RIGHI, R. D. R. A context-aware spontaneous mobile social network. In: **2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)**. [S.l.: s.n.], 2015. p. 85–92.

NISSENBAUM, H. Privacy as contextual integrity. **Wash. L. Rev.**, HeinOnline, v. 79, p. 119, 2004.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 414–454, First 2014. ISSN 1553-877X.

PREUVENEERS, D.; BERBERS, Y. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In: **Proceedings of the 2007 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2007. (SAC '07), p. 1165–1170. ISBN 1-59593-480-4. Disponível em: <http://doi.acm.org/10.1145/1244002.1244255>.

REGO, P. A. L. **Applying Smart Decisions, Adaptive Monitoring and Mobility Support for Enhancing Offloading Systems**. Tese (Doutorado) — Universidade Federal do Ceará, 2016.

SAEED, A.; WAHEED, T. An extensive survey of context-aware middleware architectures. In: **Electro/Information Technology (EIT), 2010 IEEE International Conference on**. [S.l.: s.n.], 2010. p. 1–6. ISSN 2154-0357.

SALVIATO, T. P. Suporte a aplicações sensíveis ao contexto no cenário do sistema brasileiro de televisão digital. Universidade Federal do Espírito Santo, 2012.

SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; BUYYA, R. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 369–392, First 2014. ISSN 1553-877X.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **Pervasive Computing, IEEE**, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1536-1268.

SCHILIT, B. N.; THEIMER, M. M. Disseminating active map information to mobile hosts. **IEEE network**, IEEE, v. 8, n. 5, p. 22–32, 1994.

SHERCHAN, W.; JAYARAMAN, P. P.; KRISHNASWAMY, S.; ZASLAVSKY, A.; LOKE, S.; SINHA, A. Using on-the-move mining for mobile crowdsensing. In: **Mobile Data Management (MDM), 2012 IEEE 13th International Conference on**. [S.l.: s.n.], 2012. p. 115–124.

SHIRAZ, M.; GANI, A.; KHOKHAR, R.; BUYYA, R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. **Communications Surveys Tutorials, IEEE**, v. 15, n. 3, p. 1294–1313, Third 2013. ISSN 1553-877X.

SUO, H.; LIU, Z.; WAN, J.; ZHOU, K. Security and privacy in mobile cloud computing. In: **2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.: s.n.], 2013. p. 655–659. ISSN 2376-6492.

TOCH, E. Crowdsourcing privacy preferences in context-aware applications. **Personal Ubiquitous Comput.**, Springer-Verlag, London, UK, UK, v. 18, n. 1, p. 129–141, jan. 2014. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s00779-012-0632-0>.

VIANA, W. **Mobility and Context-awareness for Personal Multimedia Management: CoMMediA**. Tese (Theses) — Université Joseph-Fourier - Grenoble I, fev. 2010. Financé par CAPES-Brésil. Disponível em: <https://tel.archives-ouvertes.fr/tel-00499550>.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Modelos e processos para o desenvolvimento de sistemas sensíveis ao contexto. **André Ponce de Leon F. de Carvalho, Tomasz Kowaltowski.(Org.). Jornadas de Atualização em Informática**, p. 381–431, 2009.

WEISER, M. The computer for the 21st century. **SIGMOBILE Mob. Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 3, n. 3, p. 3–11, jul. 1999. ISSN 1559-1662. Disponível em: <http://doi.acm.org/10.1145/329124.329126>.

YURUR, O.; LIU, C.; SHENG, Z.; LEUNG, V.; MORENO, W.; LEUNG, K. Context-awareness for mobile sensing: A survey and future directions. **Communications Surveys Tutorials, IEEE**, PP, n. 99, p. 1–1, 2014. ISSN 1553-877X.

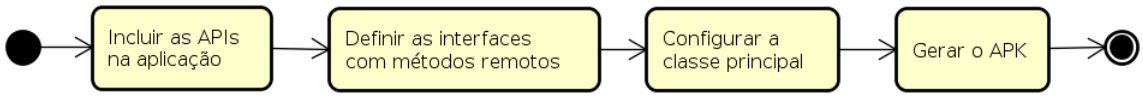
APÊNDICE A – PROCESSO DE CONFIGURAÇÃO

Este apêndice descreve todo o processo de configuração da solução proposta. O desenvolvedor precisa definir algumas configurações para integrar o serviço do COP em conjunto com a aplicação móvel, tanto na parte cliente quanto na parte servidora. São necessárias três configurações: da aplicação, do serviço no servidor e do serviço no cliente. A seguir, são descritas cada uma delas.

Configuração da Aplicação

A Figura 36 apresenta um diagrama, que contém os 4 passos que o desenvolvedor deve seguir para configurar a aplicação com o COP.

Figura 36 – Diagrama de Configuração da Aplicação



Fonte: Elaborado pelo autor

No COP, o processo de configuração inicia com a integração da API em conjunto com o projeto da aplicação. Em seguida, são realizadas diversas configurações diretamente no código fonte da aplicação, por meio de marcações de código necessárias pelo MpOS.

No segundo passo, o desenvolvedor deve definir as Interfaces que possuem métodos candidatos à realização de *offloading* por meio da marcação *@Remotable* no código-fonte. Como os filtros contextuais podem serem executados em nuvem, eles são implementados dentro desses métodos remotos. A Listagem 6 apresenta um exemplo de como seria essa Interface.

Listagem 6 – Exemplo de Interface de Métodos Remotos

```
1 public interface IMyInterface {
2   @Remotable
3   public String read();
4 }
```

Para ilustrar como acontece a implementação da Interface de métodos remotos, bem com um exemplo de implementação de um filtro local e um filtro remoto, é apresentado a

Listagem 7, que é uma extensão da Listagem 1. Nesse exemplo, a Classe implementa a Interface da Listagem 6 e foi implementado um filtro contextual para consultar as tuplas que representam a temperatura do ambiente. No filtro local, o avaliador de expressão retorna "`true`" apenas se a tupla analisada tiver o valor do campo "`Values`" maior que "`25.0`". Assim, se a temperatura do ambiente for maior do que 25ºC, a tupla analisada é retornada para a aplicação. No filtro remoto, o avaliador retorna "`true`" se no campo "`IdApp`" tiver valor "`br.casa.app`" e o campo "`Values`" tiver valor maior ou igual a "`25.0`". Assim, se a temperatura do ambiente for maior ou igual a 25ºC e a aplicação identificada for "br.casa.app", a tupla é retornada para a aplicação.

Listagem 7 – Exemplo de Filtro Contextual da Solução Proposta

```
1  public class MyInterface implements IMyInterface {
2    ...
3    Pattern p = (Pattern) new Pattern();
4    p.addField("ContextKey", "context.ambient.temperature");
5    List<Tuple> result = (ArrayList) COPManager.read(p, f);
6    ...
7    IFilter.Stub f = new IFilter.Stub() {
8
9      @Override
10     public boolean localFilter(Tuple tuple) throws RemoteException {
11       NumberListVariable lVar = new NumberListVariable("Values", 0);
12       Expression vExp = gt(lVar, new NumberConstant(25.0));
13       return Evaluator.eval(tuple, vExp);
14     }
15
16     @Override
17     public boolean remoteFilter(Tuple tuple) throws RemoteException {
18       StringListVariable idApp = new StringListVariable("IdApp");
19       NumberListVariable values = new NumberListVariable("Values", 0);
20
21       Expression idAppExp = eq(idApp, new StringConstant("br.casa.app"));
22       Expression valuesExp = gteq(values, new NumberConstant(25.0));
23
24       Expression fExp = and(idAppExp, valuesExp);
25       return Evaluator.eval(tuple, fExp);
26     }
27   }
28   ...
29 }
```

Após definir as interfaces que possuem métodos candidatos à realização do *offloading* de processamento, o terceiro passo é integrar o processo de inicialização do COP e MpOS na

aplicação móvel. Esta etapa é feita por meio da customização da Classe principal da aplicação Android (i.e., a primeira activity). O desenvolvedor precisa definir as variáveis de instância que são do tipo das interfaces que possuem métodos marcados para realizar a operação de *offloading*. Assim, o desenvolvedor precisa marcar as variáveis de instância com a marcação *@Inject*, definindo na propriedade dessa marcação, o tipo concreto que será instanciado dinamicamente por essas variáveis. Assim, esse processo de marcação permitirá que o MpOS intercepte todos os métodos invocados, a partir dessas variáveis que foram dinamicamente instanciadas. A Listagem 8 apresenta um exemplo de como seria essa classe principal.

Listagem 8 – Exemplo de Classe Principal

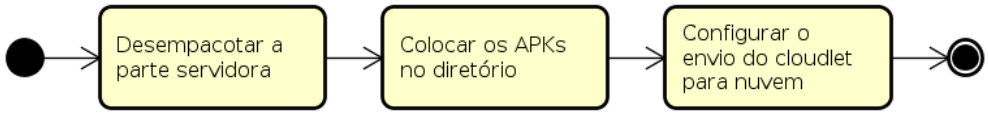
```
1  @MposConfig
2  public class MainActivity extends Activity {
3    @Inject(MyInterface.class)
4    IMyInterface myInterface;
5
6    COPManager cop;
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     ...
12     Mpos.getInstance().start(this, this);
13     cop = new COPManager(this, getPackageName());
14     cop.start(this);
15     ...
16   }
17   ...
```

No último passo da configuração do COP na aplicação, o desenvolvedor precisa gerar um APK da aplicação desenvolvida, o qual será armazenada em um diretório no servidor, para que o MpOS consiga realizar as operações de *offloading* requisitadas.

Configuração do Serviço no Servidor

O processo de configuração do COP no lado servidor é constituído por 3 passos, que são ilustrados no diagrama da Figura 37. O primeiro passo é a implantação de todos os arquivos necessários para o funcionamento do MpOS e do COP em um *cloudlet/cloud* (i.e., os executáveis Java, os diretórios que servem para armazena os APKs e um arquivo de configuração).

Figura 37 – Diagrama de Configuração do Servidor



Fonte: Elaborado pelo autor

No segundo passo, o desenvolvedor deve colocar os APKs das aplicações móveis gerados em uma pasta que será responsável por guardar todos os APKs que utilizam o MpOS. Como no COP é possível ter *cloudlets* enviando dados contextuais para uma nuvem, o desenvolvedor precisa definir em um arquivo de configuração o endereço da máquina servidora na nuvem no último passo.

A Listagem 9 apresenta um exemplo deste arquivo de configuração. Na propriedade *prop.server.ip.cloud* é definido o endereço da máquina que executa na nuvem. Se caso for definido um valor para essa propriedade, o *cloudlet* deve enviar os dados contextuais para esse endereço. Caso contrário, os dados não são enviados.

Listagem 9 – Arquivo de Configuração do Servidor

```
1 prop.server.ip.cloud = 200.129.43.208
```

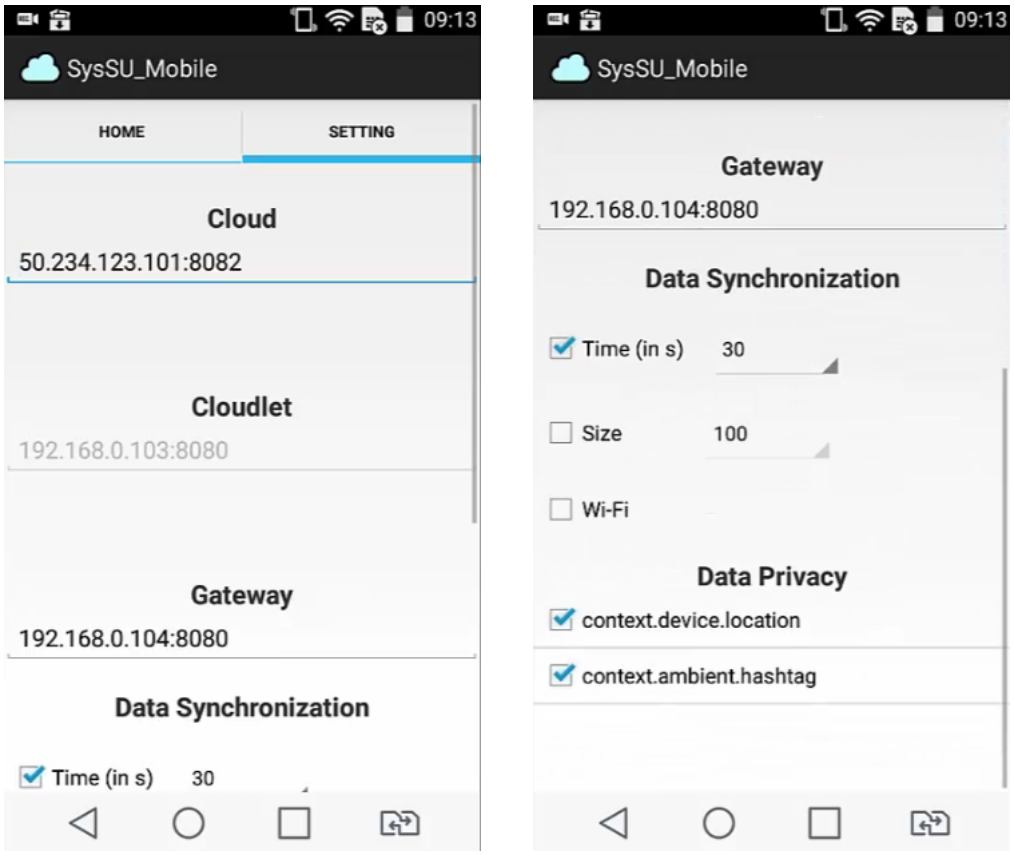
Configuração do Serviço no Cliente

Uma vez que o serviço COP está instalado no dispositivo móvel e é iniciado pelo usuário, são necessárias algumas configurações. No serviço do COP existe um componente responsável pela descoberta de *cloudlet*. Se caso não exista nenhum *cloudlet* em funcionamento, o usuário pode configurar o endereço da nuvem para o qual ele deseja enviar os dados contextuais. No serviço, o usuário pode configurar o *gateway* da preferência dele. Para isso, o usuário deve entrar com o endereço do *cloudlet* descoberto e o COP irá armazenar o endereço *Media Access Control* (MAC) do servidor.

No serviço também é possível configurar as políticas de sincronização que serão utilizadas durante o envio dos dados contextuais para o ambiente remoto. Para a privacidade dos dados contextuais que serão enviados, o usuário deve marcar quais dados são considerados privados para ele e que não devem ser enviados para o ambiente remoto. Como os dados

contextuais são representados a partir da *ContextKey* de cada CAC, todos os CACs que serão utilizados pelas aplicações devem estar instalados no dispositivo móvel. A Figura 38 apresenta *screenshots* da configuração do serviço COP.

Figura 38 – *Screenshots* da Configuração do Serviço COP



(a) (b)

Fonte: Elaborado pelo autor

Uma vez iniciado o servidor do MpOS, bem como todos os serviços necessários, o usuário pode executar a aplicação móvel e realizar a operação de *offloading*, quando possível.