Uma Proposta de Extensão de um Middleware de Aquisição de Contexto para Suporte a Crowdsensing com Privacidade

Francisco A. A. Gomes
Universidade Federal do Ceará – UFC
Campus do Pici, s/n, CEP 60451-970
Fortaleza – Ceará – Brasil
franciscoanderson@great.ufc.br

Lincoln S. Rocha
Universidade Federal do Ceará – UFC
Campus do Pici, s/n, CEP 60451-970
Fortaleza – Ceará – Brasil
lincoln@great.ufc.br

Fernando A. M. Trinta
Universidade Federal do Ceará – UFC
Campus do Pici, s/n, CEP 60451-970
Fortaleza – Ceará – Brasil
fernandotrinta@great.ufc.br

ABSTRACT

Mobile devices such as smarthphones have become a common tool in our daily routine. Mobile Applications (a.k.a. apps) have increasingly demanded access to contextual information, such as environment and system data, as well as user profile. These apps adapt themselves according this context data in order to improve their services. Mobile Applications with this behavior are known as context-aware applications. Several software infrastructures have been created to help the development of mobile context-aware applications, but most of them do not store the historic of contextual information, once mobile devices are resource constrained (computing and memory capabilities). They are not built taking into account the privacy of contextual information either, due the fact that apps can expose contextual data, without without user consent. This paper addresses this topic by extending an existing middleware platform called LoCCAM to store and process large volumes of contextual information generated from several mobile devices (crowdsensing); and the definition of privacy policies that avoid dissemination of unauthorized contextual information.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software ; C.5.3 [**Computer System Implementation**]: Microcomputers

General Terms

Management, Performance, Languages

Keywords

Cloud, Context-Aware, Mobile Device, Middleware, Offloading

1. INTRODUÇÃO

Dispositivos móveis, tais como *smartphones*, *tablets* e *smartwatches*, dotados de uma série de sensores tornaram-se populares e comuns no nosso dia a dia. Acessar, a partir destes dispositivos, aplicativos de redes sociais, *e-mails*, bate-papos virtuais, a qualquer hora e em qualquer lugar, faz parte do cotidiano de muitas pessoas. Essas aplicações acessam, cada vez mais, informações contextuais do ambiente, do sistema e do usuário para se adaptarem às mudanças e oferecerem serviços relevantes. Aplicativos dotados desse comportamento são conhecidos como sensíveis ao contexto [10].

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 [1, 2, 5, 10, 12, 16]. Essas infraestruturas proveem funcionalidades que facilitam a aquisição e processamento de informações 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 informações de contexto como a temperatura ambiente e a umidade relativa do ar para recomendar aos corredores, 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 variação dessas "variáveis de contexto" e notificá-la quando limiares pré-estabelecidos fossem atingidos, desencadeando assim as recomendações.

Apesar da grande quantidade de infraestruturas, muitas não armazenam o histórico das informações contextuais de interesse das aplicações, bem como não suportam o armazenamento de um grande volume de dados gerado pelos usuários e nem possuem componentes que realizem inferências com os esse dados [11, 15]. Isso decorre do fato dos dispositivos móveis ainda serem computacionalmente limitados. Assim, não é possível esse armazenamento, o que se faz necessário em várias situações (e.g., um sistema de recomendação que sugere filmes a um usuário baseado no histórico dos filmes assistidos por ele). Além disso, muitas infraestruturas não são construídas levando em consideração a privacidade das

informações contextuais, o que pode acarretar divulgação pública de informações contextuais do usuário sem a sua devida anuência. Assim, é necessário que a infraestrutura possua políticas de privacidade para que o usuário possa tornar pública ou não as suas informações contextuais para outros usuários.

Com o objetivo de mitigar os problemas apresentados, este trabalho de mestrado propõe a extensão de um *middleware* existente, denominado LoCCAM (*Loosely Coupled Context Acquisition Middleware*) [9], incluindo funcionalidades que lhe permitem lidar com: (i) o armazenamento e processamento de grandes volumes de informações contextuais geradas a partir de vários dispositivos móveis (*crowdsensing*) [6, 7]; (ii) a definição de políticas de privacidade que visão restringir a divulgação e difusão de informações contextuais de cada usuário do sistema. Para isso, a técnica de *offloading* [3], na qual dados e/ou processamento podem ser descarregados do dispositivo móvel para um ambiente com maior capacidade computacional (e.g., servidor remoto ou um ambiente de nuvem), é empregada.

O restante deste artigo encontra-se organizado como segue. Na próxima seção é apresentado o *middleware* LoCCAM. Detalhes sobre a técnica de *offloading* é apresentado na Seção 3. A Seção 4 descreve a proposta de extensão do LoCCAM, assim como estudos de caso que serão realizados para validação do trabalho. Por fim, a Seção 5 apresenta os trabalhos futuros e conclui o artigo. O papel de cada autor nesse artigo é este: o primeiro autor é o aluno, o segundo autor é o co-orientador e o último autor é o orientador.

2. O MIDDLEWARE LOCCAM

O LoCCAM é um *middleware* que dá suporte ao desenvolvimento e execução de sistemas sensíveis ao contexto na plataforma Android [9]. Esse *middleware* intermedia de forma adaptativa a aquisição de contexto, utilizando os sensores físicos, lógicos e virtuais presentes no próprio dispositivo móvel. Os sensores físicos são os que adquirem informações diretamente de sensores fornecidos pelo dispositivo móvel (e.g., temperatura e umidade relativa do ar). Os sensores lógicos são os que utilizam informações geradas por outros sensores (físicos ou virtuais) para realizar inferências e gerar uma nova informação contextual (e.g., sensor que faz uma relação entre a temperatura e umidade relativa do ar para gerar uma informação de risco de desidratação). Os sensores virtuais são os que adquirem informações contextuais a partir de outros sistemas de software ou serviços (e.g., um componente que acessa um serviço web meteorológico para fornecer informações climáticas).

No LoCCAM, os elementos responsáveis pela captura das informações contextuais (i.e., sensores) são os Componentes de Aquisição de Contexto (CAC). Essas informações são disponibilizadas 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 informação contextual. O segundo caso segue um modelo de interação do tipo *publish-subscribe*, onde as aplicações são notificadas quando as informações de contexto de interesse são inseridas no espaço de tuplas. A camada responsável pela aquisição de contexto e a camada de aplicação são desacopladas. Desse modo, a obtenção das informações contextuais ocorre de forma transparente, na qual as aplicações não necessitam conhecer a origem das informações contextuais para utilizá-las. 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 informações de contexto.

O *middleware* pode ser dividido em duas partes principais: o *framework* CAM (Context Acquisition Manager) e o módulo SysSU (*System Support for Ubiquity*) [8]. Uma visão geral da arquitetura do LoCCAM é apresentada na Figura 1. 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* assume a função de, efetivamente, executá-la.

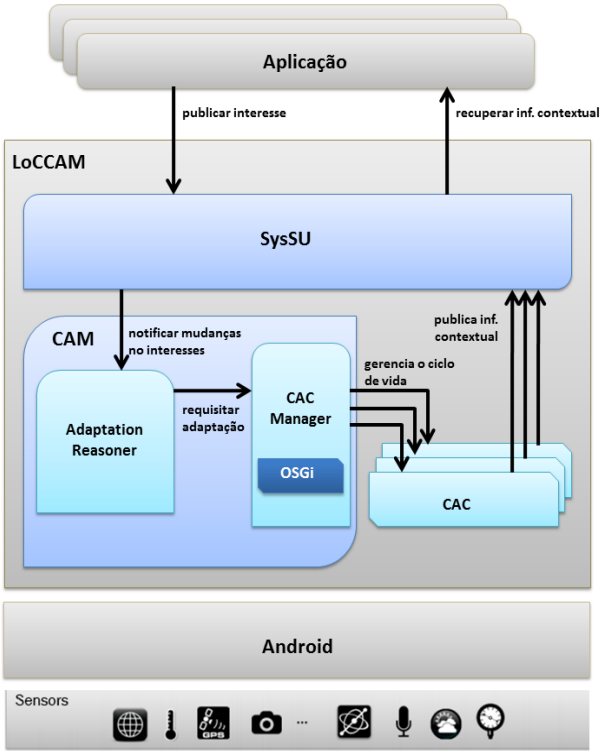

Figura 1: Arquitetura do LoCCAM (Adaptado de [4])

O módulo SysSU, explicado em mais detalhes na próxima seção, é um intermediador entre o LoCCAM, as aplicações e os CACs. As aplicações e os CACs registram no SysSU seus interesses em um determinado tipo de informação contextual. Uma vez que o SysSU toma conhecimento desses interesses, ele notifica o *Adaptasion Reasoner*, que por sua vez, estabelece quais adaptações deverão ser feitas na camada de

aquisição de contexto (i.e., quais CACs serão ativados ou desativados) 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*. Mais detalhes sobre o SysSU são apresentados na Seção 2.1.

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

2.1 O Módulo SysSU

O SysSU é uma infraestrutura que tem por objetivo dar suporte a interação espontânea em ambientes ubíquos [8]. Para isso, o SysSU utiliza um modelo de coordenação desacoplado e uma arquitetura flexível que permite lidar com a heterogeneidade de plataformas de desenvolvimento e execução dos sistemas ubíquos [8]. Entre as funcionalidades do módulo, é permitido que informações contextuais, no formato de tuplas (Definição 1), sejam armazenadas e acessadas de maneira síncrona e assíncrona no espaço de tuplas (i.e., um conjunto finito e não ordenado de tuplas).

Definição 1 (Tupla). *Uma tupla é uma sequência de n elementos $t = \langle (n_0, v_0), (n_1, v_1), ..., (n_{n-1}, v_{n-1}) \rangle$, onde cada campo (n_i, v_i), um par nome (n_i) e valor (v_i), é único e $\forall (n_j, v_j), (n_k, v_k) \in t, j \neq k$ temos que $n_j \neq n_k$.*

Na forma de acesso síncrono, as aplicações requisitam as informações de contexto que lhes interessam e ficam bloqueadas até que: (i) o espaço de tuplas retorne as informações solicitadas; ou (ii) um determinado período de tempo transcorra (i.e., *timeout*). A forma assíncrona é baseada em notificação de eventos (*publish-subscribe*). A segunda forma de acesso é utilizada nos casos em que a aplicação pode esperar até o momento em que a tupla que contém a informação contextual de interesse esteja disponível no espaço de tuplas.

Como mencionado anteriormente, toda a interação entre as aplicações e o LoCCAM ocorre por meio do SysSU. É nele que todos os CACs publicam as informações contextuais sensoreadas. Também é através dele que as aplicações e/ou CACs podem realizar consultas, síncronas ou assíncronas.

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

Entretanto, 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. No LoCCAM, as tuplas armazenadas no SysSU possuem um formato padrão e são representadas como descrito na Definição 2.

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

A estrutura apresentada em (1) caracteriza uma leitura da informação de contexto "temperatura do ambiente" no formato de tupla do LoCCAM.

$$t = \langle (\texttt{contextkey}, \text{“}\texttt{context.ambient.temp}\text{”}), (\texttt{source}, \text{“}\texttt{physical}\text{”}), (\texttt{values}, \text{“}\texttt{25}\text{”}), (\texttt{accuracy}, \text{“}\texttt{0}\text{”}), (\texttt{timestamp}, \text{“}\texttt{239459060969}\text{”}) \rangle \quad (1)$$

O *middleware* permite duas formas de seleção das informações contextuais no espaço de tuplas do SysSU: (i) casamento de padrão (*pattern matching*); e (ii) casamento de padrão com filtro de tuplas. No casamento de padrão, 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 é retornada. 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). Caso a mesma aplicação tenha interesse em ler tuplas cujo valor da temperatura ambiente seja igual a 25 °C, o padrão (3) deve ser utilizado.

$$p = \langle (\texttt{contextkey}, \text{“}\texttt{context.ambient.temp}\text{”}) \rangle \quad (2)$$

$$q = \langle (\texttt{contextkey}, \text{“}\texttt{context.ambient.temp}\text{”}), (\texttt{values}, \text{“}\texttt{25}\text{”}) \rangle \quad (3)$$

Os filtros de tuplas permitem estabelecer critérios mais refinados para a seleção de tuplas. Eles são compostos por um padrão de tupla, utilizado para o casamento de padrão, 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 (4) pode ser utilizado. Observe que a primeira parte do filtro de tupla consiste em um padrão de tupla ((`contextkey`, "`context.ambient.temp`")) e a segunda parte uma expressão lógica que restringe o limite dos valores

da temperatura a serem recuperados (`values` > 25).

$$f = \langle\langle(\texttt{contextkey}, \text{“}\texttt{context.ambient.temp}\text{”})\rangle, \quad (4)$$
$$\texttt{values} > 25\rangle$$

A Listagem 1 apresenta um trecho de código que mostra como o filtro de tuplas (padrão + expressão lógica) descrito em (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 24 e é utilizado para consultar as tuplas que contiverem o campo cujo nome é ”`ContextKey`” e o valor seja “`context.ambient.temp`”. 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 2 até 17, o campo que será avaliado nessa tupla está na linha 6 (“`Values`”) e o valor na linha 7 (maior que “25.0”).

```
1  ...
2  IFilter.Stub filter = new IFilter.Stub() {
3
4    @Override
5    public boolean filter(Tuple tuple) throws RemoteException {
6      NumberListVariable values = new NumberListVariable("Values", 0);
7      Expression exp = gt(values, new NumberConstant(25.0));
8
9      try {
10       return Evaluator.eval(tuple, exp);
11     } catch (EvaluationException e) {
12       e.printStackTrace();
13     }
14
15     return false;
16   }
17 }
18 ...
19
20 ArrayList<Tuple> result = null;
21 try {
22   String name = "ContextKey";
23   String value = "context.ambient.temp";
24   Pattern pattern = (Pattern) new Pattern().addField(name, value);
25   result = (ArrayList) SysSUManager.read(pattern, filter);
26 } catch (RemoteException e) {
27   e.printStackTrace();
28 }
29 ...
```

Listagem 1: Exemplo de filtro contextual

O LoCCAM foi projetado para manter apenas o estado mais recente das informações contextuais. Dessa maneira, o SysSU mantém apenas uma única tupla de um tipo de informação contextual (e.g., a última localização capturada pelo dispositivo ou a última leitura da temperatura ambiente). Porém, dependendo dos requisitos da aplicação, um histórico de leituras de informações contextuais faz-se necessário para melhorar a precisão dos algoritmos de inferência. Entretanto, armazenar no próprio dispositivo móvel essas informações é uma solução proibitiva, visto o grande volume de leituras geradas para cada tipo de sensor. Nesse sentido, uma solução seria enviar essas informações contextuais, de forma oportunística, para algum ambiente centralizado capaz de processar esse grande volume de dados gerado pelos diversos dispositivos móveis que compõem o sistema. Assim, além de amenizar os problemas de armazenamento, seria possível realizar inferências mais precisas levando em consideração o contexto global (i.e., o somatório dos contextos de cada dispositivo móvel) e consultas mais refinadas.

3. OFFLOADING

O paradigma de *Mobile Cloud Computing* [14] (MCC) busca mitigar problemas relacionados as limitações de recursos dos dispositivos móveis empregando a combinação de tecnologias em diversas áreas, tais como: computação em nuvem, computação móvel e redes sem fio. O paradigma de MCC concentra-se nos benefícios que podem ser alcançados por esses dispositivos, quando se delega uma operação de armazenamento ou processamento de dados, para um ambiente de execução remoto com maior capacidade computacional. Assim, os dispositivos móveis tentem a economizar recursos (e.g., consumo de energia e espaço de armazenamento) e aumentar o seu desempenho computacional. Essa técnica de delegação de armazenamento e processamento é referenciada na literatura como *cyber-foraging* ou *offloading* [3]. O *offloading* pode ser executado em máquinas virtuais na nuvem pública ou em qualquer máquina que esteja na mesma rede local dos dispositivos móveis.

Existem duas formas principais de *offloading*: (i) de processamento; (ii) de dados. O *offloading* de processamento é a entrega de um processamento computacional do dispositivo móvel para outro ambiente de execução (e.g., *notebook*), a fim de prolongar a vida útil da bateria e aumentar a capacidade computacional. O *offloading* de dados tem por objetivo melhorar o envio de dados entre o dispositivo móvel e a nuvem, assim retira o armazenamento do dispositivo e envia para um ambiente com maior capacidade de armazenamento e permite que esses dados possam ser enviados de volta para o dispositivo móvel.

4. PROPOSTA DE EXTENSÃO

Com o objetivo de mitigar os problemas relacionados ao armazenamento de grandes volumes de informação de contexto e a privacidade dessas informações, este trabalho propõe a extensão do *middleware* LoCCAM, incluindo funcionalidades que lhe permitem lidar com: (i) o armazenamento e processamento de grandes volumes de informações contextuais geradas a partir de vários dispositivos móveis (*crowdsensing*); (ii) a definição de políticas de privacidade que visão restringir a divulgação e difusão de informações contextuais de cada usuário do sistema.

Para lidar com o problema de armazenamento, é empregada a técnica de *offloading* de dados. Nesse caso, as informações contextuais geradas por todos os dispositivos do sistema são enviadas, seguindo uma política pré-estabelecida, para um ambiente centralizado de armazenamento e processamento em nuvem, denominado de SysSU Cloud. Para garantir a identificação da origem das informações de contexto, lidar com questões de relógio em ambiente distribuído, e privacidade, uma extensão no formato da tupla do LoCCAM foi proposto (veja a Definição 3).

Definição 3 (Tupla no SysSU Cloud). *Uma tupla no SysSU Cloud é representada por 9 elementos, os 5 primeiros da Definição 2,* $t = \langle(\texttt{contextkey}, \text{“?”}), (\texttt{source}, \text{“?”}), (\texttt{values}, \text{“?”}), (\texttt{accuracy}, \text{“?”}), (\texttt{timestamp}, \text{“?”}), (\texttt{iddevice}, \text{“?”}), (\texttt{idapp}, \text{“?”}), (\texttt{visible}, \text{“?”}), (\texttt{globaltimestamp}, \text{“?”})\rangle$,

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 da informação contextual, se pública ou privada; e* `globaltimestamp` *representa o instante, em milissegundos, em que a informação contextual é inserida no SysSU Cloud.*

A política de privacidade está relacionada com a visibilidade da informação contextual. O usuário pode estabelecer explicitamente quais informações ele autoriza serem enviadas para o SysSU Cloud ou não. Na prática, o valor do campo `visible` da tupla deve ser atribuído 1, para indicar que a visibilidade é pública ou 0, caso a visibilidade seja privada. A tupla (5) caracteriza uma leitura da informação de contexto "temperatura do ambiente", do dispositivo móvel com identificação "`0424033418`" e aplicação com identificação "`br.ufc.great.temp`" no novo formato.

$$
\begin{aligned}
t = \langle &(\texttt{contextkey}, \text{“}\texttt{context.ambient.temp}\text{”}), \\
&(\texttt{source}, \text{“}\texttt{physical}\text{”}), \\
&(\texttt{values}, \text{“}\texttt{25}\text{”}), \\
&(\texttt{accuracy}, \text{“}\texttt{0}\text{”}), \\
&(\texttt{timestamp}, \text{“}\texttt{239459060969}\text{”}) \\
&(\texttt{iddevice}, \text{“}\texttt{0424033418}\text{”}) \\
&(\texttt{idapp}, \text{“}\texttt{br.ufc.great.temp}\text{”}) \\
&(\texttt{visible}, \text{“}\texttt{1}\text{”}) \\
&(\texttt{globaltimestamp}, \text{“}\texttt{239459780932}\rangle
\end{aligned} \tag{5}
$$

A comunicação entre as aplicações e o *middleware* acontece por meio de uma API. Essa API provê uma forma padrão de acesso ao LoCCAM que executa como um serviço de sistema dentro do dispositivo móvel. Na extensão proposta, o acesso as informações contextuais acontece de maneira transparente, podendo ser no dispositivo móvel (SysSU local) ou na nuvem (SysSU Cloud). Para garantir esse acesso transparente, a API utiliza um mecanismo de descoberta do local e direciona a execução dos filtros contextuais dessas informações para o serviço local (no diapositivo móvel) ou o serviço remoto (na nuvem). Para a execução desses filtros na nuvem, o LoCCAM utiliza uma infraestrutura que da suporte ao *offloading* de processamento[2], assim, quando oportuno (i.e., baseado nas condições de rede e no consumo de bateria), os filtros de tupla são descarregados para nuvem e executados sobre as informações de contexto que estão no SysSU Cloud.

Na extensão do *middleware*, foi modificada a interface `IFilter`, de tal forma que o desenvolvedor da aplicação deve implementar dois filtros contextuais: o que será executado local e o que será executado remoto. Esses filtros podem ser diferentes, pois os algoritmos remotos, como serão executados na nuvem, podem ser mais complexos que os algoritmos locais. A Listagem 2 apresenta como pode ser implementado um filtro local e o filtro remoto. Nesse exemplo, foi implementado um filtro contextual para consultar as tuplas que representam a temperatura do ambiente. Caso a consulta seja local, as tuplas analisadas serão as do espaço de tuplas do SysSU. Caso a consulta for remota, as tuplas analisadas serão as do SysSU Cloud. No filtro local, o avaliador de expressão retorna "`true`" apenas se a tupla analisada tiver o valor do campo "`Values`" maior que "`10.0`". No filtro remoto, o avaliador retorna "`true`" se no campo "`IdApp`" tiver valor "`br.ufc.great.app`" e o campo "`Values`" tiver valor maior ou igual a "30.0".

[2]https://github.com/ufc-great/mpos

```
...
IFilter.Stub filterTemp = new IFilter.Stub() {

  @Override
  public boolean localFilter(Tuple tuple) throws RemoteException {
    NumberListVariable listVariable = new NumberListVariable("Values", 0);
    Expression valuesExp = gt(listVariable, new NumberConstant(10.0));

    try {
      return Evaluator.eval(tuple, valuesExp);
    } catch (EvaluationException e) {
      e.printStackTrace();
    }

    return false;
  }

  @Override
  public boolean remoteFilter(Tuple tuple) throws RemoteException {
    StringListVariable idApp = new StringListVariable("IdApp");
    NumberListVariable values = new NumberListVariable("Values", 0);

    Expression idAppExp = eq(idApp, new StringConstant("br.ufc.great.app"));
    Expression valuesExp = gteq(values, new NumberConstant(30.0));

    Expression finalExp = and(idAppExp, valuesExp);

    try {
      return Evaluator.eval(tuple, finalExp);
    } catch (EvaluationException e) {
      e.printStackTrace();
    }

    return false;
  }
}
...

ArrayList<Tuple> result = null;
try {
  Pattern pattern = (Pattern) new Pattern().addField("ContextKey", "context.ambient.temp");
  result = (ArrayList) SysSUManager.read(pattern, filterTemp);
} catch (RemoteException e) {
  e.printStackTrace();
}
...
```

Listagem 2: Exemplo de filtro contextual da proposta

4.1 Arquitetura

Uma visão geral da arquitetura do LoCCAM com o SysSU Cloud pode ser visto na Figura 2. No lado mobile da arquitetura, foi acrescentado um banco de dados NoSQL, orientado a documentos, para funcionar como uma espécie de *cache* local das informações de contexto até que o processo de *offloading* ocorra.

O componente `Synchronizer` é responsável por recuperar as informações contextuais do banco de dados no dispositivo móvel (*cache* local) e enviar ao SysSU Cloud por meio de um *webservice* do tipo RESTful. O `Synchronizer` utiliza estratégias diferentes para o envio das informações contextuais para o SysSU Cloud. Três estratégias de envio encontram-se implementadas, a saber: por tempo, por tamanho, orientado a conexão WiFi. A estratégia de envio por tempo significa que de tempos em tempos (e.g., a cada 30 segundos) as tuplas são enviadas ao SysSU Cloud. A estratégia de envio por tamanho implica que se o banco de dados atingir uma quantidade pré-estabelecida de tuplas, elas serão enviadas. E por fim, a estratégia de envio orientação a conexão WiFi define que se um conexão de de rede WiFi for estabelecida, as tuplas são envidas. Por padrão é utilizado a estratégia de envio por tempo. Sempre que as tuplas forem enviadas

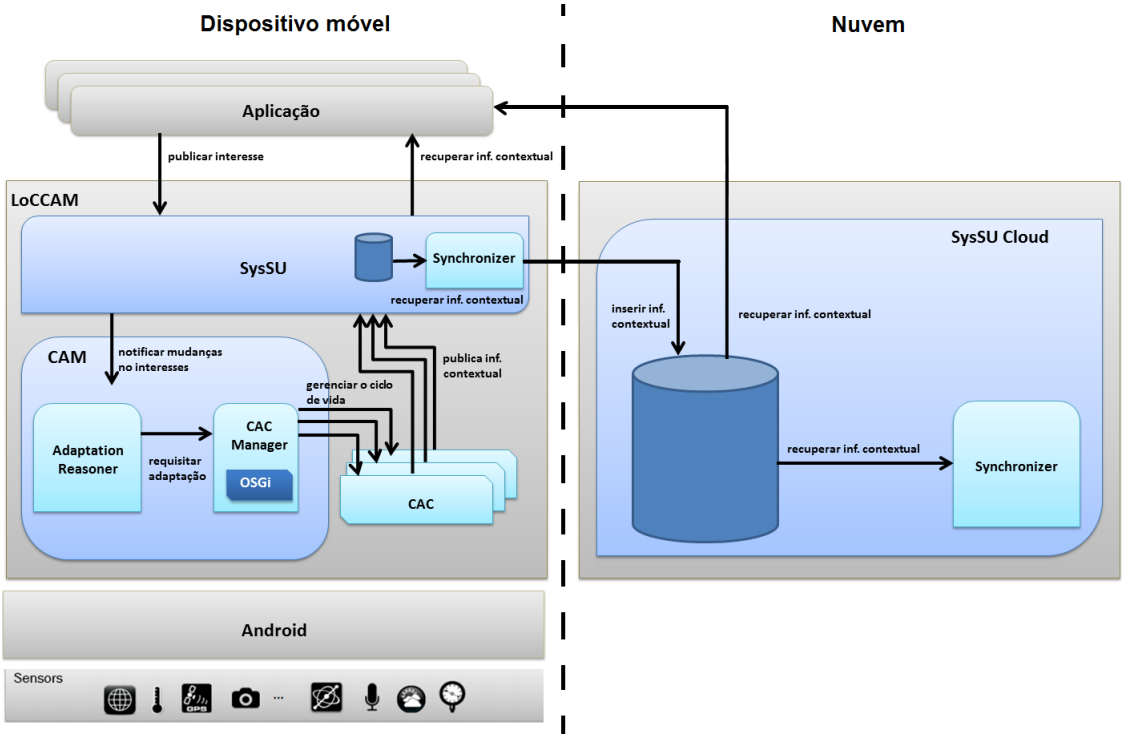

Figura 2: Arquitetura do LoCCAM com a extensão do SysSU Cloud

para o SysSU Cloud, o banco de dados local é esvaziado para evitar o acúmulo de de dados no dispositivo móvel.

A versão inicial do SysSU Cloud foi desenvolvida para executar em um *cloudlet*[3]. Uma política de privacidade baseada na visibilidade da informação contextual (campo `visible` da tupla) e no grau de confiança do *cloudlet* conectado foi estabelecida. Essa política permite que informações contextuais com visibilidade privada só sejam difundidas para um *cloudlet* confiável, denominado de *personal gateway*.

Por exemplo, o usuário deseja que a posição geográfica dele seja privada para outros usuários, mas permite que a temperatura do ambiente capturada pela infraestrutura seja pública. Assim, no momento da sincronização, se o dispositivo móvel não estiver conectado a um *personal gateway*, apenas as tuplas públicas serão enviadas, caso contrário, se o dispositivo estiver conectado a um *personal gateway* todas as informações contextuais serão enviadas, independente da sua visibilidade.

O SysSU Cloud possui um banco de dados NoSQL orientado a documentos, similar ao que executa no dispositivo móvel, e que tem por objetivo a persistência de um grande volume de informações contextuais dos dispositivos móveis que compõem o sistema. Além disso, o SysSU Cloud possui um componente `Synchronizer`, que é responsável pelo envio de dados para outro *cloudlet* ou para uma nuvem pública. Nesse trabalho, é previsto a execução do SysSU Cloud em um nuvem pública, então, as tuplas poderão ser enviadas dos *cloudlets* para a nuvem ou diretamente dos dispositivos para a nuvem pública. Contudo, é importante mencionar que as informações privadas armazenadas nos *personal gateways* não são enviadas para a nuvem pública.

[3]Um *cloudlet* pode ser visto como um ambiente de execução (e.g. *desktop* e *notebook*) que interage com os dispositivos móveis dentro de uma mesma rede local sem fio, provendo meios para a realização de *offloading* [13]

4.2 Estudos de Caso

De modo a validar a proposta deste trabalho e ilustrar os ganhos com o armazenamento das informações contextuais dos dispositivos móveis pelo *middleware*, são previstos dois estudos de caso: (i) um aplicativo de recomendação e (ii) um aplicativo que identifica o estado de movimento de uma pessoa (e.g., parado, andando e correndo). O primeiro estudo de caso visa mostrar como informações contextuais podem ser usadas para sugerir itens baseado no interesse do usuário de aplicação de compartilhamento de fotos. Já o segundo estudo tem por objetivo mostrar como o histórico das informações contextuais pode melhorar a inferência de uma aplicação móvel.

4.2.1 Sistema de Recomendação

Este experimento tem como proposta a implementação de uma aplicação de compartilhamento de fotografias, semelhante ao Instagram [4]. Usuários podem compartilhar fotos e marcá-las com *hashtags*, de modo a facilitar a busca entre usuários. Além disso, fotos também são marcadas com a localização e o instante (data/hora) em que as mesmas são capturadas pelo dispositivo móvel do usuário. Uma vez que o SysSU Cloud guardará as informações de vários usuários, a aplicação irá sugerir *hashtags* de fotos com contexto semelhante, i.e., com local e instante semelhantes ao de novas fotos. Este cenário pode ser melhor explicado por um exemplo

[4]https://instagram.com

mais concreto. Imagine um usuário durante a abertura dos jogos olímpicos do Rio 2016. Em um dado instante, o usuário utiliza a aplicação para tirar fotos da cerimônia. Uma vez que outros usuários também podem estar usando a aplicações, compartilhando fotos e marcando-as com *hashtags* como #rio2016 e #openingceremony, estas *hashtags* podem ser recomendadas a outros usuários que também estejam no Estádio Olímpico na festa de abertura.

Para testar a aplicação, serão realizados experimentos controlados, onde um conjunto de dados será gerado artificialmente, e a recomendação de *hashtags* será avaliada. Além desta avaliação, a aplicação será ofertada a um número de usuários ainda a ser definido. Estes usuários avaliarão a ferramenta em relação à qualidade das *hashtags* recomendadas por meio de questionários.

4.2.2 Inferência

O experimento da inferência do estado de uma pessoa utilizará o histórico das informações contextuais de localização e acelerômetro. De acordo com horário do dia, a aplicação indicará se a pessoa está andando, correndo ou parado. Para avaliar a aplicação, 5 pessoas utilizaram o aplicativo durante uma semana e anotaram a indicação do estado (i.e., andando, correndo ou parado) de acordo com a aplicação e o estado realmente em que se encontram. Ao final da semana, cada pessoa responderá um questionário sobre o que achou da coerência da indicação da aplicação. Nesse experimento, os resultados devem ser melhores ao final da semana, pois a quantidade de informação contextual é maior, assim o algoritmo de inferência deve ser mais preciso.

5. CONCLUSÃO E TRABALHOS FUTUROS

Este artigo apresentou uma proposta de extensão do *middleware* LoCCAM com suporte ao armazenamento e processamento de grandes volumes de informações contextuais e um mecanismo para definição de políticas de privacidade. Para lidar com os problemas de armazenamento, a técnica de *offloading* foi empregada utilizando um ambiente de *cloudlet*. Já os problemas de privacidade foram resolvidos modificando a estrutura tuplas que representam as informações contextuais e criando o conceito de *personal gateways*, que são *cloudlets* confiáveis.

Esse trabalho de mestrado tem a defesa prevista para Julho de 2016 e como trabalho futuro alguns pontos precisam ser trabalhados, tais como: (i) a execução do SysSU Cloud em uma nuvem, para que as informações contextuais presentes nos *cloudlets* sejam enviadas para um local único, permitindo uma capacidade de armazenamento ainda maior, assim como de processamento dos filtros contextuais; e (ii) a implementação e avaliação dos estudos de caso como prova de conceito do funcionamento da extensão do LoCCAM.

6. REFERÊNCIAS

[1] S. Buthpitiya, F. Luqman, M. Griss, B. Xing, and A. Dey. Hermes – a context-aware application development framework and toolkit for the mobile environment. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 663–670, March 2012.

[2] D. Carlson and A. Schrader. Dynamix: An open plug-and-play context framework for android. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 151–158, Oct 2012.

[3] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.

[4] P. A. S. Duarte, F. M. Barreto, F. A. A. Gomes, W. C. Viana, and F. A. M. Trinta. Critical: A configuration tool for context aware and mobile applications. *In: 39th Annual International Computers, Software & Applications Conference*, 1:159–168, jul 2015.

[5] M. Ferroni, A. Damiani, A. Nacci, D. Sciuto, and M. Santambrogio. coda: An open-source framework to easily design context-aware android apps. In *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*, pages 33–38, Aug 2014.

[6] R. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, November 2011.

[7] N. Haderer, R. Rouvoy, and L. Seinturier. A preliminary investigation of user incentives to leverage crowdsensing activities. In *2nd International IEEE PerCom Workshop on Hot Topics in Pervasive Computing (PerHot)*, pages 199–204, San Diego, United States, Mar. 2013. IEEE Computer Society.

[8] F. Lima, L. Rocha, P. Maia, and R. Andrade. A decoupled and interoperable architecture for coordination in ubiquitous systems. In *Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on*, pages 31–40, Sept 2011.

[9] M. E. F. Maia, A. Fonteles, B. Neto, R. Gadelha, W. Viana, and R. M. C. Andrade. Loccam - loosely coupled context acquisition middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 534–541, New York, NY, USA, 2013. ACM.

[10] D. Preuveneers and Y. Berbers. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1165–1170, New York, NY, USA, 2007. ACM.

[11] J. Punjabi, S. Parkhi, G. Taneja, and N. Giri. Relaxed context-aware machine learning midddleware (rcamm) for android. In *Intelligent Computational Systems (RAICS), 2013 IEEE Recent Advances in*, pages 92–97, Dec 2013.

[12] A. Saeed and T. Waheed. An extensive survey of context-aware middleware architectures. In *Electro/Information Technology (EIT), 2010 IEEE International Conference on*, pages 1–6, May 2010.

[13] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, Oct 2009.

[14] M. Shiraz, A. Gani, R. Khokhar, and R. Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys Tutorials, IEEE*, 15(3):1294–1313, Third 2013.

[15] E. Williams and J. Gray. Contextion: A framework for developing context-aware mobile applications. In *Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle*, MobileDeLi '14, pages 27–31, New York, NY, USA, 2014. ACM.

[16] O. Yurur, C. Liu, Z. Sheng, V. Leung, W. Moreno, and K. Leung. Context-awareness for mobile sensing: A survey and future directions. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.