Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

ContextF: framework de apoio a construção de aplicações sensíveis ao contexto

Marcus Henrique Barbosa

JUIZ DE FORA
AGOSTO, 2013

ContextF: framework de apoio a construção de aplicações sensíveis ao contexto

Marcus Henrique Barbosa

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Sistemas de Informação

Orientador: Eduardo Barrére

JUIZ DE FORA

AGOSTO, 2013

ContextF: framework de apoio a construção de aplicações sensíveis ao contexto

Marcus Henrique Barbosa

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Eduardo Barrére
D.Sc. em Engenharia de Sistemas e Computação - COPPE/UFRJ

Marcelo Ferreira Moreno
D.Sc. em Computação - PUC-Rio

Eduardo Pagani Julio
M.Sc. em Computação - UFF

JUIZ DE FORA
AGOSTO, 2013

Resumo

Contexto é o meio pelo qual as aplicações tem a capacidade de saber da existência de características relacionadas as atividades desempenhadas por usuários, equipamentos ou mesmo outras aplicações. Nesse cenário o desenvolvimento de novas aplicações que podem adaptar seu comportamento de acordo com as variações do ambiente tem crescido. Contudo, essas aplicações necessitam de um alto grau de controle sobre todos os contextos e saber como utilizá-los de forma eficaz pode ser algo complicado. Mas a partir do momento em que a aplicação possa delegar todo esse controle e se preocupar apenas com as funcionalidades que agregam maior valor, tem-se a necessidade do desenvolvimento de novas ferramentas para fornecer esse tipo de apoio. Nesta monografia é abordada a construção do ContextF, um *framework* que utilizando uma arquitetura composta de eventos e componentes desacoplados que reagem a eles, espera-se a criação de uma API simplificada e uma arquitetura facilmente extensível.

Palavras-chave: Contexto, dispositivos móveis, *framework*.

Abstract

Context is the means by which applications have the ability to know the existence of characteristics related to activities performed by users, devices or even other applications. In this scenario the development of new applications that can adapt their behavior according to the variations of the environment has grown. However, these applications need a high degree of control over all contexts and how to use them effectively can be tricky. But from the moment that the application can delegate all this control and only worry about the features that add greater value, it has been necessary the development of new tools to provide such support. This monograph is addressed to build ContextF, a framework that using an architecture composed of events and decoupled components that react to them, it is expected to create a simplified API and architecture easily extensible.

Keywords: Context, mobile devices, framework.

Agradecimentos

A todos os meus parentes, pelo encorajamento e apoio.

Ao professor Eduardo Barrére pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Sumário

1 Introdução

Atualmente, a tendência para computação ubíqua onde dispositivos, agentes de *software* e serviços devem se integrar transparentemente para cooperarem no apoio aos objetivos de seus usuários, em qualquer lugar, a qualquer momento (WEISER, 1991), tornou a sensibilidade ao contexto um grande subconjunto desse tipo de computação devido as vantagens obtidas da sua capacidade em perceber as variações do ambiente (ADDLESEE et al., 2001) (IPINA et al., 2002).

Dispositivos modernos com conectividade, poder de processamento e capacidade de utilizar sensores, representam um meio onde aplicações que necessitem perceber as variações do ambiente podem ser desenvolvidas. Contudo, essas aplicações demandam uma grande quantidade de código que deve ser escrito para interagir com sensores, lidar com problemas de intermitência de conectividade e limitada capacidade de transferência, uso eficaz dos recursos do dispositivo e armazenamento, gerando um alto custo de desenvolvimento. Alguns *frameworks* foram desenvolvidos (DEY; ABOWD; SALBER, 2001) (CARRIZO et al., 2008) (BARDRAM, 2005), porém, aplicações sensíveis ao contexto desenvolvidas com eles ficam acopladas, o que dificulta, por exemplo, a reutilização de funcionalidades comuns entre aplicações.

1.1 Objetivos

Criar o ContextF, um *framework* que gerencia o processo de contextualização para um determinado recurso multimídia. Com o intuito de desenvolver um *framework* adaptável e genérico, podendo ser utilizado em qualquer tipo de aplicação para dispositivos móveis, é importante também: desenvolver componentes desacoplados que executam a extração de informações do contexto; a sincronização com um servidor de contexto remoto; o armazenamento quando a sincronização não acontece devido a problemas com a rede; o monitoramento do estado da rede que identifica quando há disponibilidade para prosseguir com o processo de sincronização das informações armazenadas.

1.2 Estrutura do trabalho

No Capítulo 2 Seção 2.1, o conceito de computação ubíqua é apresentado e como ela pode ser afetada pelas variações do ambiente. Na Seção 2.2, é apresentado o conceito de contexto bem como os processos que podem ser utilizados para derivar novos contextos através de informações básicas, e as aplicações sensíveis ao contexto que através dele conseguem alterar seu comportamento para prover serviços de forma adequada aos seus usuários. Na Seção 2.3, é apresentada a arquitetura orientada a eventos bem como as suas propriedades e características que permitem que o ContextF seja altamente desacoplado e extensível. No Capítulo 3 Seção 3.1, é apresentada a visão geral da arquitetura do servidor de contexto e descreve, brevemente, cada um dos elementos que a compõem. No capítulo 4 Seção 4.1, faz uma revisão dos mecanismos existentes de autenticação e checagem de integridade juntamente ao modelo proposto pelo ContextF. Na Seção 4.2, é apresentado o conceito de papéis e como o seu uso permite que o usuário possa atribuir regras de visibilidade aos recursos que gera. No Capítulo 5 Seção 5.1, é apresentada a arquitetura orientada a eventos do ContextF e o componente que implementa essa arquitetura, o *bus*. Na Seção 5.2, é apresentado o componente *ContextBarrier* e como ele é utilizado para o gerenciamento de *threads*. Na Seção 5.3, é apresentado o componente *TransactionRegistry* que contém informações de cada processo de contextualização. Na Seção 5.4, é apresentada a modelagem das informações do recurso e do contexto. Na Seção 5.5, é apresentado o componente de sincronização bem como o formato do XML que é enviado ao servidor de contexto. Na Seção 5.6, é apresentada uma revisão dos mecanismos de armazenamento disponíveis e o componente que os gerencia. Na Seção 5.7, é apresentado o componente de monitoramento e porque ele é importante dentro do processo de contextualização. Na Seção 5.8, é apresentado o componente de configuração e como pode ser utilizado por aplicações individuais para alterar certos aspectos do ContextF. No Capítulo 6, é apresentado o estudo de caso de uma aplicação de captura de imagens construída com o ContextF. E por fim, a conclusão do estudo sobre a criação do ContextF para aplicações sensíveis ao contexto para dispositivos móveis.

2 Fundamentação teórica

2.1 Computação ubíqua

O surgimento de novas tecnologias de computação móvel em conjunto com a popularização dos dispositivos portáteis, a computação move-se para fora do computador e torna-se pervasiva em nossa vida cotidiana. Esse novo paradigma altamente dinâmico é caracterizado pelo emprego de novos dispositivos potáteis multifuncionais e pela constante mudança no ambiente, como consequência da mobilidade do usuário é chamado de Computação Ubíqua.

Computação Ubíqua é a área que torna possível disponibilizar informações e recursos computacionais a qualquer hora, lugar e dispositivo (WEISER, 1995). Sua idéia é que os computadores estejam presentes em todo lugar de forma não intrusiva e natural operando transparentemente sobre as expectativas dos usuários.

Para tanto, esses dispositivos devem ser capazes de perceber as constantes mudanças do ambiente que envolve o usuário a fim de capturarem informações relevantes. Esse conjunto de informações é chamado de contexto.

2.2 Contexto

Contexto é a base de conhecimento que possibilita a discriminação do que é ou não relevante em um dado momento, servindo de apoio a comunicação entre a aplicação e seus usuários, melhorando a qualidade da compreensão de situações, eventos e ações.

Uma definição clássica e bastante referenciada de contexto é que "qualquer informação que caracteriza a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, um lugar ou um objeto considerados relevantes para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. O contexto é tipicamente a localização, a identidade e o estado das pessoas, grupos ou objetos físicos e computacionais."(DEY; ABOWD; SALBER, 2001).

Contexto pode ser categorizado de diversas formas como, a maneira que é gerado (PARK; LEE, 2005) :

- **Informações de contexto obtidas de sensores**: A utilização de sensores é o método fundamental para extrair informações do contexto. Essas informações podem ser extraídas de diversos tipos de sensores como localização, temperatura e movimento. Esse tipo de contexto é chamado de "contexto elementar"ou "contexto puro"devido ao fato de ser uma informação desestruturada pois é apenas uma conversão do que foi capturado pelo sensor.

- **Informações de contexto combinadas**: Contextos complexos podem ser obtidos através da combinação. Em (WILLIAMSON; BERNHARD; CHAMBERLIN, 2000), existem agregadores que produzem contextos combinados a partir de informações extraídas por sensores. Basicamente, são contextos elementares em uma estrutura bem definida. Por exemplo, a combinação de informações como data e hora, localização e a agenda do usuário, pode identificar que o mesmo encontra-se em um evento, gerando um novo contexto.

- **Informações de contexto inferidas**: Contextos complexos também podem ser inferidos a partir de contextos elementares e outros complexos. O processo de inferência de contextos necessita de uma base de conhecimento para ajudar a produzir informações que não estão presentes nos contextos elementares e combinados. Por exemplo, inferir o perfil da rede social de pessoas que estão presentes em uma foto.

- **Informações de contexto aprendidas**: Contexto pode ser obtido de técnicas de aprendizagem assim como a combinação e a inferência. Contextos aprendidos desempenham papel importante ao fornecer serviços de forma adequada as preferências de usuários individuais. Técnicas de aprendizagem de máquinas e mineração de dados permitem que novas informações de contexto possam ser descobertas como interesses do usuário e previsibilidade. Por exemplo, alterar a resolução de imagens e vídeos que são enviados para dispositivos móveis com baixa largura de banda.

Contudo, é necessária a criação de aplicações capazes de gerenciar essas in-

formações do contexto capturadas por meio de sensores, combinação, inferência e aprendizado.

2.2.1 Aplicações sensíveis ao contexto

Aplicações sensíveis ao contexto são aquelas que manipulam elementos contextuais para apoiar o usuário na execução de alguma tarefa pela adaptação de seu comportamento a partir de informações extraídas do contexto.

Diferentemente das aplicações tradicionais que agem apenas levando em consideração as entradas explicitamente fornecidas por usuários, as aplicações sensíveis ao contexto levam em consideração também as informações obtidas por meio de de sensores, mecanismos de inferência e outras fontes de informações contextuais, conforme mostra a Figura 2.1.

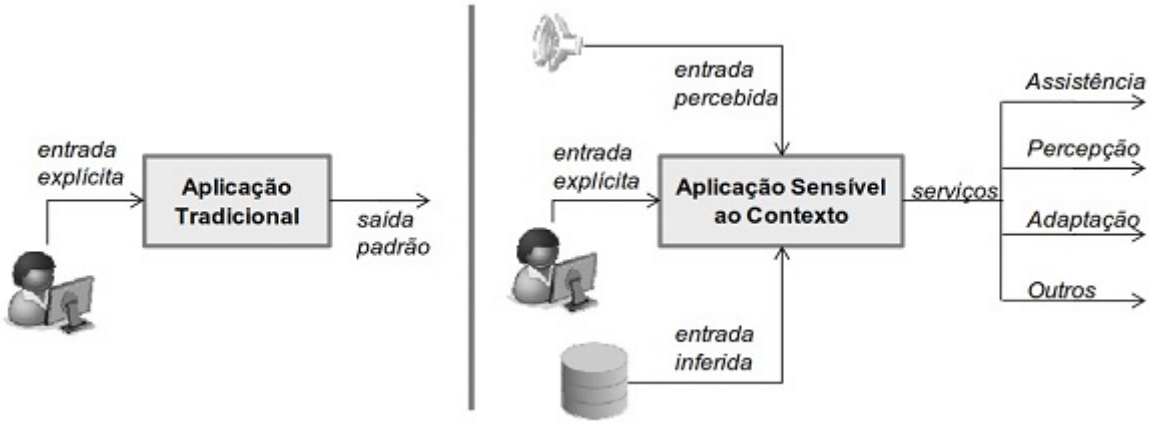

Figura 2.1: Aplicação Tradicional vs Aplicação Sensível ao contexto

Porém, ainda existem muitos desafios a serem superados ao se projetar uma aplicação sensível ao contexto, como a aquisição dos elementos contextuais a partir de fontes heterogêneas, o processamento e interpretação das informações adquiridas e a adaptação da aplicação de acordo com o contexto observado. Além disso, a aplicação deve ser crítica quanto ao tratamento da qualidade da informação obtida, tratamento de regras de segurança e privacidade e, não menos importante, quanto a desempenho.

Estão sendo desenvolvidas pesquisas que visam a geração de ferramentas e métodos específicos para o tratamento desses desafios. Essas pesquisas buscam fornecer funcionalidades básicas relativas ao gerenciamento da informação contextual de forma que as

aplicações possam fazer uso simplificando seu desenvolvimento. (VIEIRA et al., 2008) sumariza três categorias como requisitos para simplificar o desenvolvimento desse tipo de aplicação: Especificação do contexto, Gerenciamento do contexto e Uso do contexto.

2.2.2 Especificação do contexto

Por contexto possuir natureza dinâmica, torna-se impossível enumerar todas as situações que podem existir na aplicação, sendo necessária a identificação de que elementos contextuais podem determinar com precisão a situação desse conjunto e decidir as ações que devem ser executadas. (KORPIPAA et al., 2003) propõe os seguintes critérios: habilidade para descrever propriedades úteis do mundo real, possibilidade de inferência de contextos complexos e facilidade ou viabilidade de ser medido ou reconhecido automaticamente da forma mais precisa possível, para ajudar na escolha de informações relevantes do contexto.

Visando facilitar a identificação do que considerar como contexto, as informações capturadas do mesmo podem ser classificadas segundo critérios como dimensões, granularidade e periodicidade de atualização.

Quanto a dimensão, as informações contextuais podem ser classificadas pelos 5W+1H (BULCÃO; PIMENTEL, 2006) como:

- ***who***: relacionadas a identidade da entidde
- ***what***: relacionadas as atividades em que uma entidade está envolvida
- ***where***: relacionadas a localização da entidade
- ***when***: relacionadas ao contexto temporal relacionado a uma interação com a entidade
- ***why***: relacionadas a motivação por trás das ações do usuário ao executar uma tarefa em uma interação
- ***how***: relacionadas a forma como as informações contextuais são obtidas

Quanto a granularidade, as informações contextuais podem ser classificadas como básicas ou complexas (WANG et al., 2004):

- ***básicas***: obtidas automaticamente (latitude, longitude, data e hora) e por si só não são relevantes ao foco do usuário

- ***complexas***: obtidas por meio de processos de combinação, inferência ou aprendizado gerando informações mais relevantes ao foco do usuário.

Quanto a periodicidade de atualização, podem ser classificadas como estáticas ou dinâmicas (HONG et al., 2004):

- ***estáticas***: não mudam frequentemente (pontos turísticos e dados pessoais)

- ***dinâmicas***: mudam de acordo com as variações do ambiente (temperatura e localização)

2.2.3 Exemplos de aplicações sensíveis ao contexto

- **CAPTAIN: geração de diários de bordo digitais usando anotação contextual e conteúdo multimídia**: projetada para organizar dados multimídias em um dispositivo móvel e gerar documentos *web* cuja estrutura é baseada em uma ou mais trajetórias (BRAGA et al., 2011). É composto por uma aplicação móvel executada por um iPhone que faz a captura de informações durante toda interação, uma aplicação *desktop* que sincroniza com o dispositivo móvel quando conectado ao computador permitindo tratar e adicionar novas informações além de armazená-las caso não exista conexão e uma aplicação *web* que registra as informações enviadas pela aplicação *desktop* tornando públicas as trajetórias, anotações e fotos dos tripulantes.

- **Dynamic Tour Guide**: composta por agentes móveis que proveem rotas personalizadas, guia navegacional e informações ambientais, levando em consideração os diferentes tipos de turistas ou necessidade de passeio (KRAMER et al., 2005). Dessa forma, informações como localização, data e velocidade de caminhada são consideradas para prover serviços de forma adequada aos seus usuários.

2.3 Arquitetura orientada a eventos

Arquiteturas que possuem eventos como entidades de primeira classe. Sempre que um evento é disparado uma série de serviços registrados são disparados. Esse tipo de arquitetura permite que os componentes existam de forma desacoplada pois produtores e consumidores de eventos não se conhecem, delegando para a arquitetura a responsabilidade de identificá-los e fazer a ponte de comunicação. Em ambientes com recursos limitados como dispositivos móveis, os componentes executam de forma reativa, ou seja, somente serão executados quando for necessário, em vez de pró-ativamente consumir recursos em vão caso não exista nenhum processamento a ser executado.

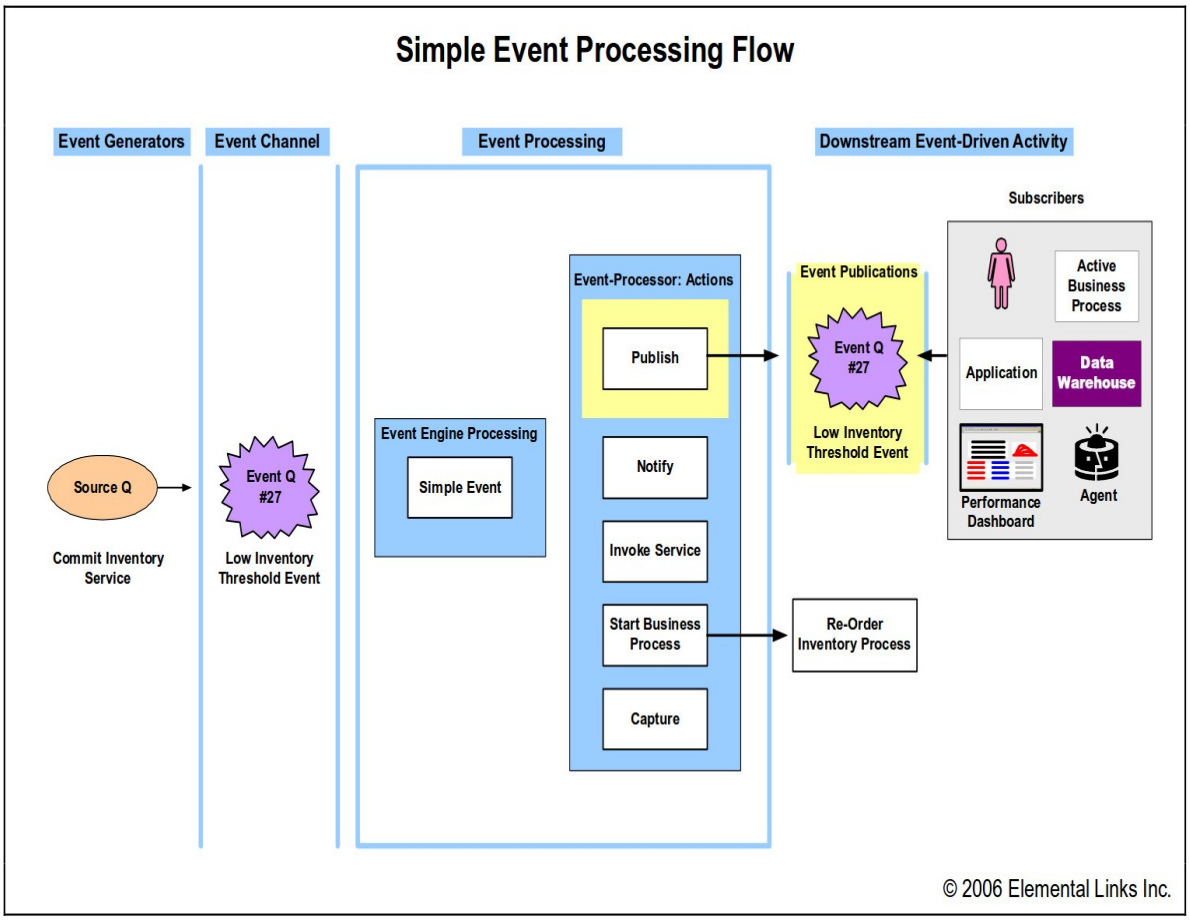

Figura 2.2: Arquitetura orientada a eventos: produtores e consumidores (MICHELSON, 2006)

2.3.1 Evento

São entidades de primeira classe que representam situações importantes dentro da aplicação que podem siginificar um problema ou uma oportunidade. Geralmente, só fazem sentido dentro da aplicação quando especificados em termos de regras de negócio. Todo evento carrega consigo as informações que originaram seu disparo, descritíveis o suficiente para que os consumidores possam processá-las idenpendentemente. Além disso, os eventos são tipados permitindo segregar produtores e consumidores.

2.3.2 Comunicação assíncrona

É o componente responsável por garantir que os eventos gerados por produtores sejam entregues as seus respectivos consumidores. Uma de suas características é o uso comunicação assíncrona, ou seja, sempre que um evento é disparado um nova *thread* é criada para entregar aos consumidores daquele evento. Assim, a aplicação não precisa interromper seu fluxo de execução.

2.3.3 Desacoplamento

O uso de eventos permite que componentes troquem informações sem possuírem uma referência direta entre eles, deixando-os totalmente desacoplados, favorecendo a evolução e a manutenibilidade do *software* impactando de forma positiva no custo do mesmo.

2.3.4 Programação reativa

O modelo de troca de informações faz com que os componentes tenham uma ação menos pró-ativa e passem a reagir como consequência do disparo de um evento. A programação reativa possibilita que os recursos sejam utilizados de forma eficaz especialmente em dispositivos móveis onde os mesmos são concorridos com outras aplicações.

3 Servidor de contexto

3.1 Arquitetura geral

Embora esta monografia foque no desenvolvimento de aplicações para dispositivos móveis, essas fazem parte de uma arquitetura maior, em desenvolvimento por outro trabalho, com o intuito de armazenar e produzir contextos que possam ser consumidos por outras aplicações. A Figura 3.1 exibe essa arquitetura a ser detalhada nas subseções.

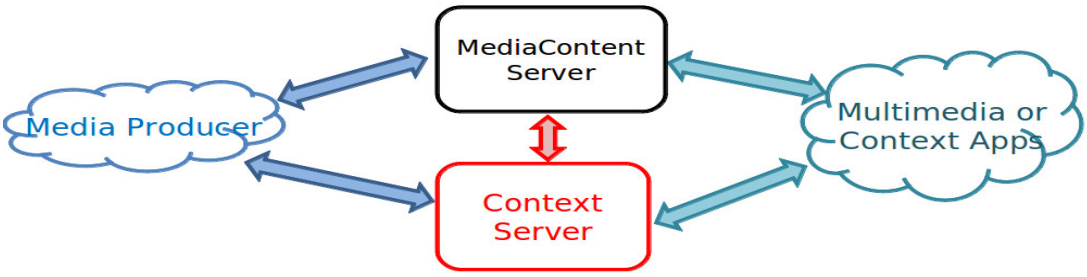

Figura 3.1: Arquitetura geral: produtores, consumidores, servidores de conteúdo e contexto

3.1.1 *Media producer*

Responsável por capturar recursos multimídias e obter informações de contexto básicas. A Figura 3.2 exibe uma visão detalhada da composição do *Media producer*. As seguintes subseções destacam cada um dos componentes envolvidos.

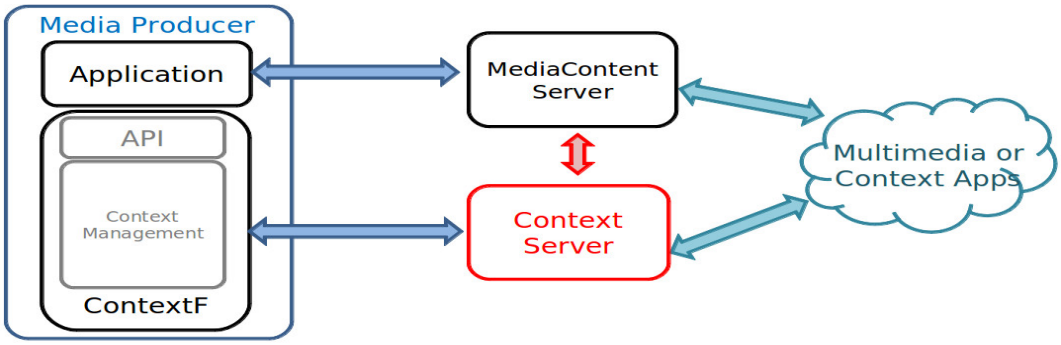

Figura 3.2: Composição do *Media producer*

- ***Application***: aplicação exibida ao usuário que o permite capturar, iniciar o processo de contextualização e enviar o recurso para o *Media content server*.

- ***ContextF: API***: é a interface de comunicação entre a aplicação e o ContextF.

- ***ContextF: Context management***: faz o gerenciamento do processo de contextualização de um determinado recurso sendo responsável por extrair informações do contexto e sincronizá-las com o *Context server*. Nota-se que, apesar de ser um produtor de recursos e contexto, a aplicação e o ContextF também atuam como consumidores.

3.1.2 *Media content server*

Servidor que contém as mídias enviadas pelo produtor. Pode ser qualquer repositório de mídias, por exemplo, as presentes em serviços como Facebook, Instagram e Google Plus ou repositórios privados pertencentes a alguma empresa.

3.1.3 *Context server*

Servidor que recebe, armazena, processa e disponibiliza as informações de contexto enviadas pelo produtor. A figura 3.3 exibe as camadas que compõem o servidor. As seguintes subseções detalham brevemente cada uma delas.

- ***Context storage service***: camada responsável por implementar a interface de comunicação entre o produtor e o servidor de contexto.

- ***Security***: camada responsável por implementar o mecanismo de segurança garantindo o cumprimento das regras de autenticação e privacidade das informações.

- ***Context transaction***: camada responsável por gerenciar toda a troca de informações entre as outras camadas e garantir a segregação dos processos de manipulação do contexto.

- ***Persistence***: camada responsável por armazenar de forma persistente as informações de contexto.

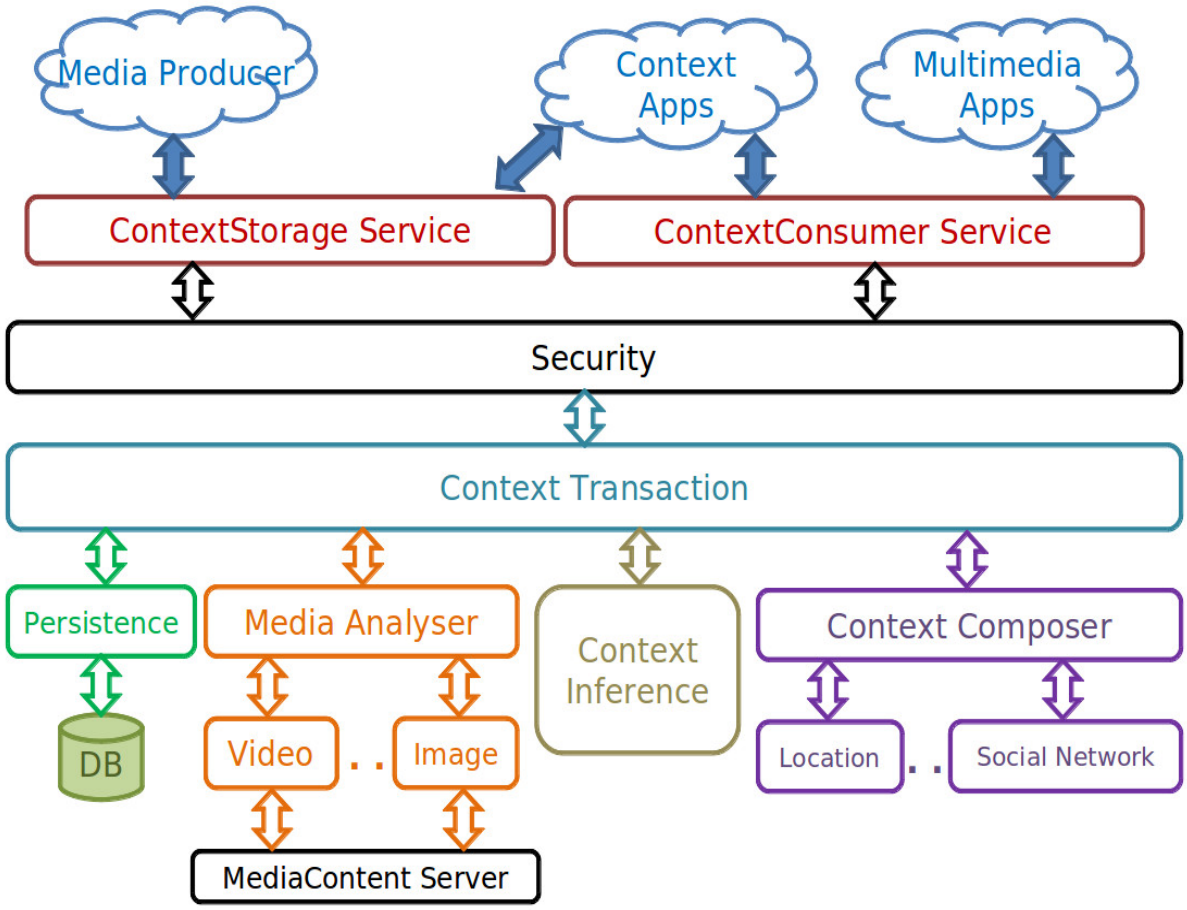

Figura 3.3: Composição do *Context server*

- ***Media analyser***: camada responsável por extrair informações de contexto a partir dos recursos enviados para o *Media content server*.
- ***Context inference***: camada responsável por descobrir outros contextos utilizando técnicas de combinação, inferência e aprendizado de máquina.
- ***Context composer***: camada responsável por transformar informações de contexto básicas como latitude, longitude ou o id do usuário de determinada rede social, em informações que agregam valor ao recurso.

3.1.4 *Multimedia or context applications*

São as aplicações que consomem informações do contexto. Elas são: outras aplicações que geram contexto; aplicações que exibem informações do contexto como serviços ou páginas web; aplicações multimídias para tv digital. A figura 3.4 exibe a composição

dessas aplicações.

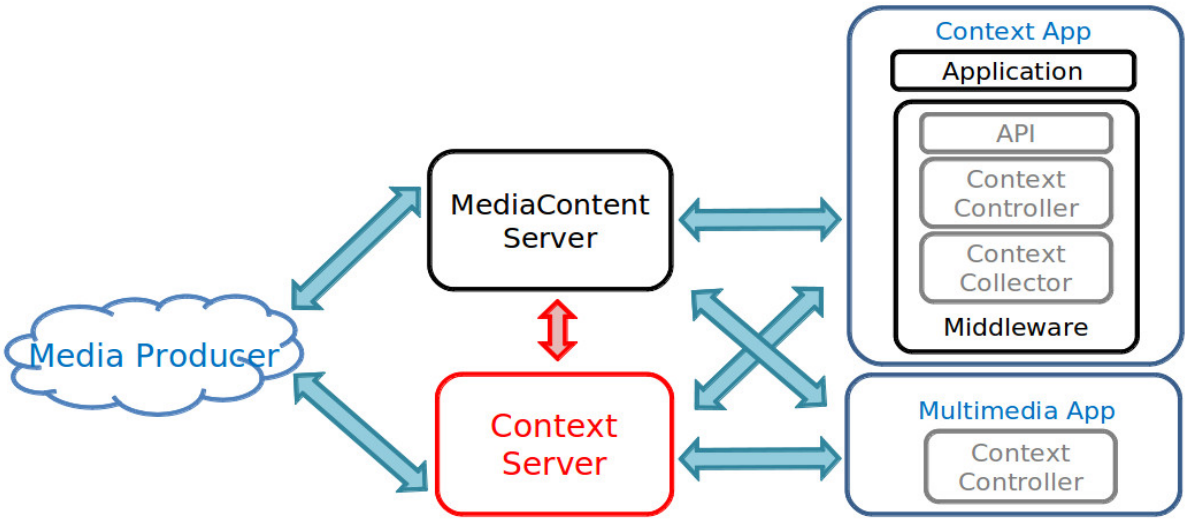

Figura 3.4: Composição das aplicações que consomem contexto

4 ContextF: Módulo de segurança

ContextF é um *framework* que permite as aplicações utilizarem de informações extraídas do contexto para adequar seu comportamento às necessidades de seus usuários. Seus principais módulos são os de autenticação e contextualização de recursos.

4.1 Autenticação

A aplicação deve obrigatoriamente estar autenticada no servidor para o envio de qualquer informação. O servidor requer que ela envie o endereço MAC do dispositivo e o ID do usuário no Google, porém outras informações complementares podem ser utilizadas.

Do ponto de vista do servidor, ContextF é considerado um cliente como qualquer outro que acessa via browser no paradigma cliente/servidor.

4.1.1 OAuth2

O OAuth2 é um mecanismo de autenticação cliente/servidor muito utilizado em redes socias como Facebook e Google Plus. Ele implementa um mecanismo de autenticação *3-legged*, ou seja, o usuário deve executar 3 passos para permitir que uma aplicação externa acesse seus dados em um determinado servidor.

Primeiramente o usuário acessa uma aplicação que deve requisitar um *token* de acesso, o usuário então é redirecionado para uma página do servidor onde ele deve fornecer seu usuário e senha e permitir que a aplicação acesse seus dados. Então, um *token* gerado é retornado a aplicação que pode ser usado para acessar os dados do usuário armazenados no servidor nas requisições subsequentes. O diagrama de sequência a seguir exibe o comportamento do mecanismo de autenticação provido pelo OAuth2.

Contudo, OAuth2 não define mecanismos para a checagem de integridade das requisições, sendo recomendado o uso de protocolos como o SSL. Como não é o foco, desta monografia, produzir mecanismos de autenticação muito robustos, ContextF define seu próprio mecanismo de autenticação e checagem de integridade.

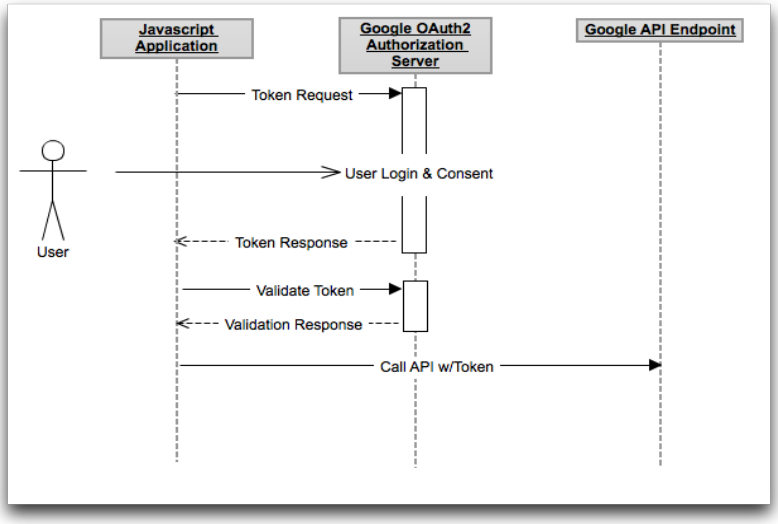

Figura 4.1: Autenticação com OAuth2

4.1.2 Mecanismo de autenticação do ContextF

O mecanismo proposto é uma adaptação do OAuth2 *3-legged* chamada *2-legged*. Basicamente, essa adaptação elimina a necessidade do usuário de ter que intervir para permitir o acesso da aplicação, pois nesse cenário a aplicação e usuário desempenham o mesmo papel, por isso o nome *2-legged*. A Figura 4.2 exibe o diagrama de sequência para o mecanismo proposto.

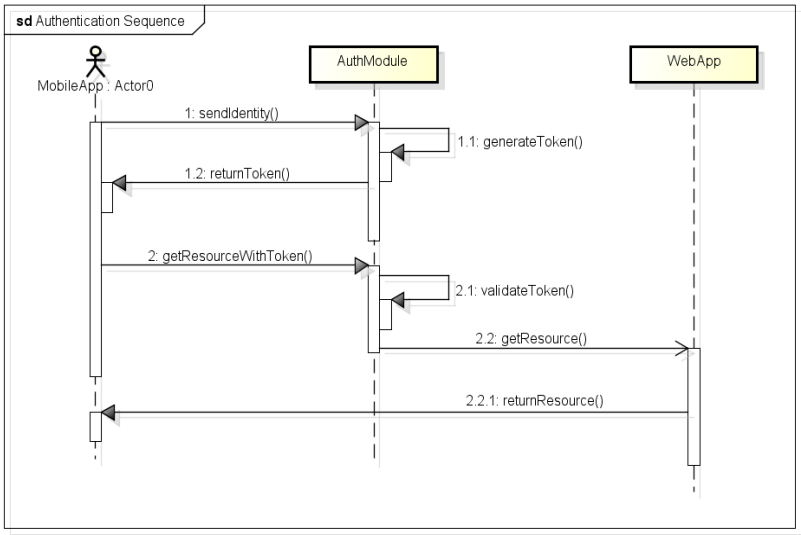

Figura 4.2: Diagrama de sequência do mecanismo de autenticação

4.1.3 Checagem de integridade

Existem algumas formas de se implementar esse tipo de integridade para requisições HTTP como é o caso do ContextF. As seguintes subseções destacam a utilização do protocolo SSL, bastante difundido nesse cenário, e o mecanismo proposto para o ContextF.

Protocolo SSL

SSL é um protocolo criptográfico muito utilizado em diversos tipos de transferência de dados como o HTTP. Seu mecanismo baseia-se no conceito de pares de chaves pública-privada utilizadas para criptografar os dados transferidos na comunicação entre aplicações, prevenindo o acesso ou alteração indevida. Dessa forma, as duas pontas da comunicação podem verificar a integridade dos dados.

Como dito, seu funcionamento é baseado no conceito de chaves pública-privada para criptografar conhecida como criptografia assimétrica. Tomando como exemplo o protocolo HTTP, a chave pública é enviada ao navegador que está acessando o servidor, estabelecendo o meio de comunicação criptografado. Quando o navegador faz uma requisição ao servidor, ele utiliza a chave pública para criptografar a mensagem. O servidor quando recebe a requisição ele utiliza a chave privada correspondente para descriptografar a mensagem da requisição. A figura 4.3 exibe o mecanismo proposto pelo SSL.

4.1.4 Mecanismo de integridade do ContextF

Como o foco desta monografia não é o estudo de meios robustos para comunicação segura entre aplicações, para fazer a checagem de integridade dos dados, foi estabelecido um mecanismo mais simples entre o ContextF e o servidor de contexto. Nesse mecanismo, são combinadas chaves utilizadas para a geração de requisições que, de alguma forma, o servidor de contexto possa checar integridade.

Assinatura

Cada requisição contém um parâmetro especial que é sua assinatura, geralmente composta por um conjunto parâmetros presentes na requisição e em uma determinada ordem,

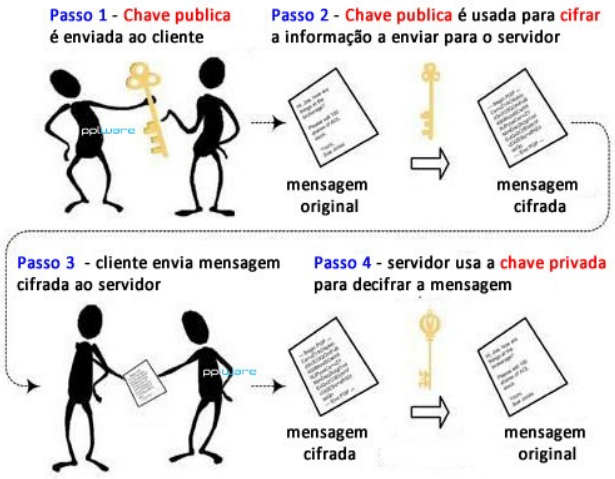

Figura 4.3: Exemplo de checagem de integridade utilizando o mecanismo de chaves pública-privada. Fonte: Grupo de Teleinformática e Informação (2011)

estabelecida previamente entre o ContextF e o servidor de contexto. Quando a requisição é recebida, o servidor gera uma assinatura, baseada no mesmo conjunto de parâmetros utilizados pelo ContextF, e compara com a recebida. Caso não sejam iguais, o servidor responde ao ContextF com o *status* 403 definido no protocolo HTTP. Apesar de simples, esse mecanismo é utilizado por grandes empresas, por exemplo, a Amazon. As Figuras 4.4 e 4.5 exibem esse esquema baseado nesse conceito de assinatura.

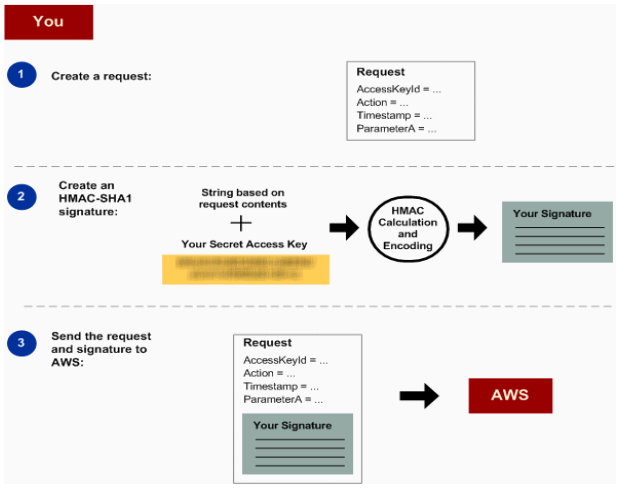

Figura 4.4: Exemplo de geração de requisições assinadas no ContextF. Fonte: http://docs.aws.amazon.com/AmazonS3/latest/dev/S3_Authentication2.html

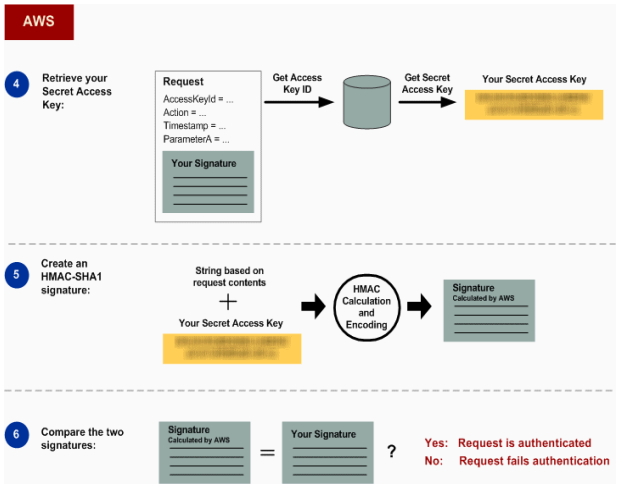

Figura 4.5: Exemplo de checagem de integridade feito pelo servidor de contexto. Fonte: http://docs.aws.amazon.com/AmazonS3/latest/dev/S3_Authentication2.html

4.2 Papéis

Com a constante evolução em termos de poder de processamento e conectividade, os dispositivos móveis estão cada vez mais presentes nas atividades diárias das pessoas seja no trabalho, em casa ou em momentos de lazer. Logo, ao capturar um recurso multimídia, é necessário identificar o papel que aquela pessoa está desempenhando no momento. Papel é um forma de segregar as informações de conttexto obtidas pelo usuário de acordo com a função que ele exerce no momento. Isso é importante, pois permite que as pessoas possam controlar o nível de privacidade daquele recurso. Por exemplo, suponha o caso em que diversas imagens foram capturadas por uma pessoa para a empresa em que trabalha, provavelmente essas imagens podem ser confidenciais e não devem ser acessadas por pessoas que não fazem parte daquele contexto, ou seja, daquela empresa. Logo, é necessário atribuir algum tipo de informação de privacidade a essas imagens e o papel pode ser utilizado para isso.

O papel pode estar incluido dentro de algum outro contexto, como a empresa no exemplo acima, mas não necessariamente. Por padrão ContextF e o servidor de contexto definem dois papéis básicos, público e o privado.

- **Público**: quando associado a algum recurso, significa a ausência de controle de privacidade permitindo que qualquer um possa acessar.

- **Privado**: quando associado a algum recurso, significa que o controle de privacidade

deve permitir que somente o dono pode acessar.

4.2.1 Modelo simples

Nesse modelo as permissões estão associadas diretamente ao papel. Contudo, o controle sobre as permissões dos recursos individualmente é menor, pois como estão associados ao papel e qualquer alteração, todos serão afetados, ou seja, não seria possível permitir que somente um determinado recurso pudesse ser visualizado por qualquer pessoa. O diagrama de classes 4.7 exibe esse modelo.

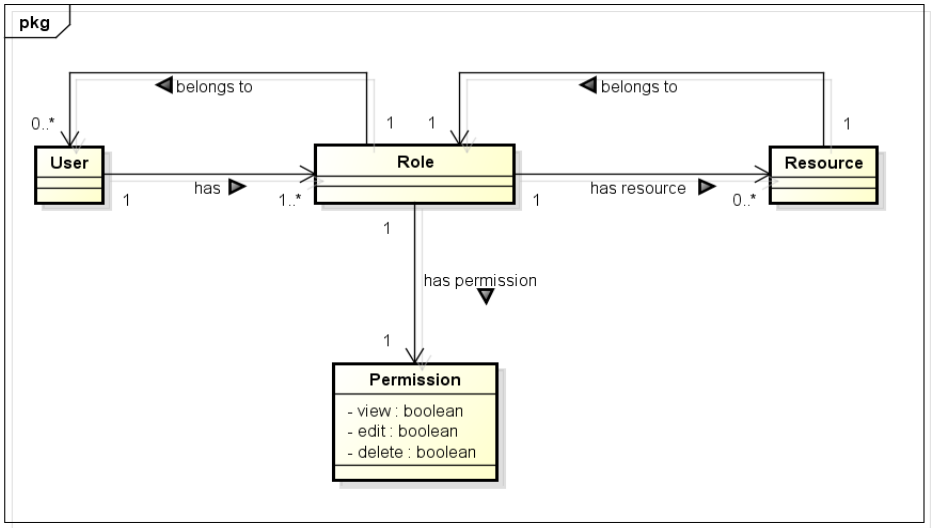

Figura 4.6: Permissões associadas diretamente ao papel

4.2.2 Modelo robusto

Diferente do modelo simples, no robusto o controle é maior, pois as permissões em vez de estarem relacionadas diretamente ao papel, agora estão relacionadas ao relacionamento entre papel e recurso possibilitando que as permissões possam ser alteradas, individualmente, recurso a recurso conforme indica o diagrama 4.7.

4.2.3 Atribuindo o papel

ContextF no momento de sua inicialização, faz uma requisição ao servidor de contexto pedindo a lista de papéis disponíveis para o usuário acessando a aplicação. A aplicação a qualquer momento pode obter essas informações e exibi-las. Então, quando um recurso

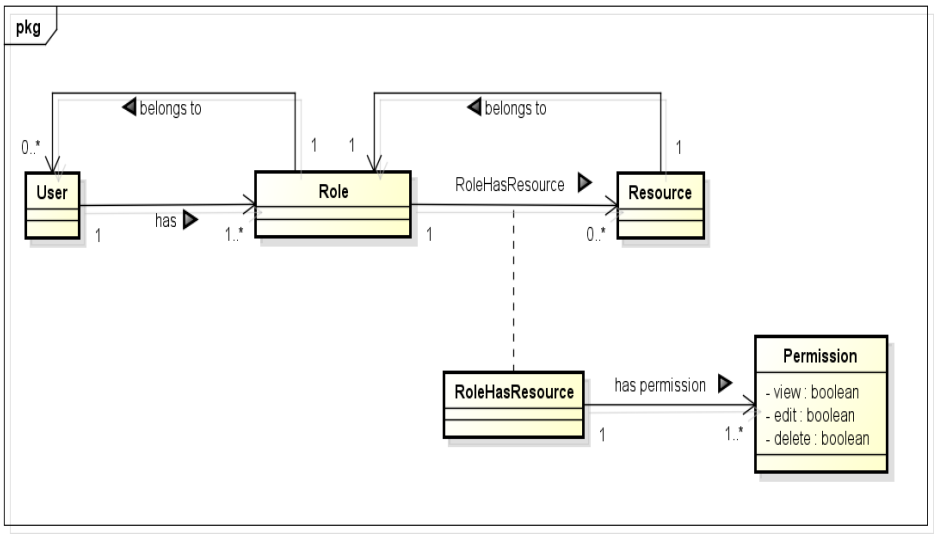

Figura 4.7: Permissões associadas ao relacionamento entre papel e recurso

é capturado, algum papel dessa lista deve ser selecionado e informado ao ContextF para que ele inclua essa informação ao sincronizar com o servidor de contexto no processo de contextualização.

5 ContextF: Módulo de contextualização

É o conjunto de etapas pelas quais as informações do contexto são obtidas, sincronizadas com o servidor de contexto ou armazenadas no próprio dispositivo quando a sincronização não ocorreu devido a problemas de conexão com a Internet. Além disso, monitora o estado da rede, reiniciando o processo de sincronização das informações armazenadas quando há conexão.

5.1 Arquitetura orientada a eventos e o processo de contextualização

O módulo é baseado em uma arquitetura orientada a eventos implementada pelo *ContextBus* que gerencia toda comunicação entre os componentes de extração e agregação de informações do contexto, controle de transação, sincronização com servidores remotos, armazenamento das informações obtidas e monitoramento da rede detalhados nas seções a seguir.

5.1.1 *ContextBus*

Baseado no padrão de comunicação *publish-subscribers*. Nesse padrão o *bus* atua como um meio de transporte para levar os eventos postados aos seus respectivos *subscribers*, os componentes que executam algum processamento baseado nesses eventos. Desse modo, os componentes que postam os eventos não precisam conhecer aqueles que tratam esses eventos e vice versa, garantindo o desacoplamento entre eles. A figura 5.1 mostra o esquema geral do *bus*.

Contudo, aplicações para dipositivos móveis que utilizam a plataforma Android, devem ser compatíveis com uma série de requisitos de gerenciamento de recursos definidos por ela. Um dos principais requisitos é a correta utilização de *threads* de forma que operações como acessar a rede ou executar algum tipo de processamento demorado, não

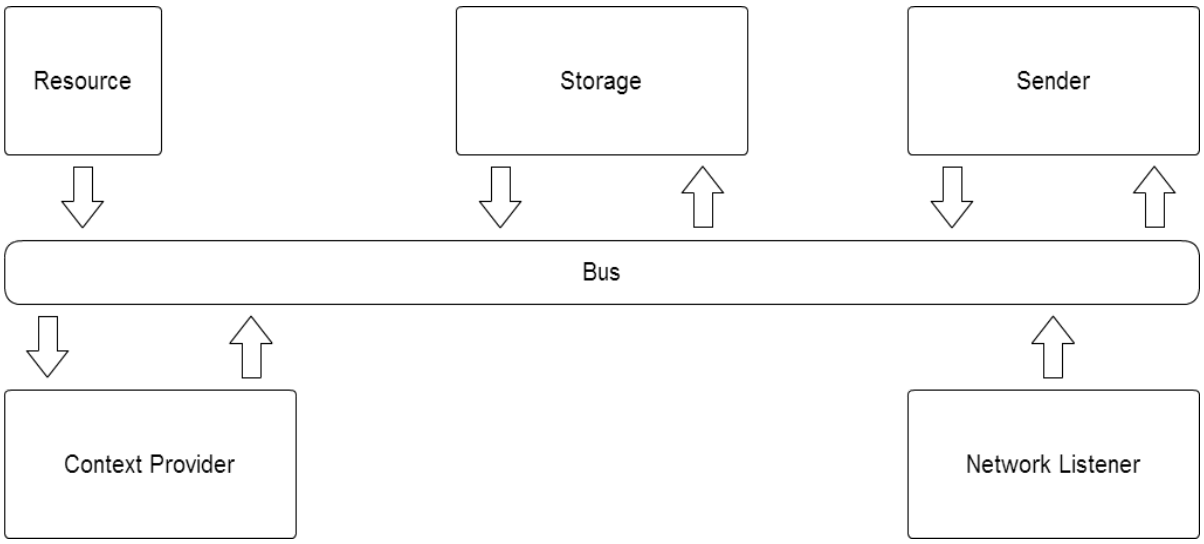

Figura 5.1: Interação entre o *bus* e os componentes do processo de contextualização

executem na *thread* principal da aplicação, evitando que o usuário experimente situações de travamento enquanto utiliza. Além disso, as *threads* devem ser gerenciadas de forma correta e utilizadas somente quando necessário, pois é muito provável que o dispositivo esteja executando outras aplicações ao mesmo tempo.

Threads

Thread é o meio pelo qual uma aplicação pode executar em mais de uma linha de execução em paralelo. Contudo, existe uma série de problemas inerentes a sua utilização. Um desses problemas que afeta o ContextF é o da ordem de execução das *threads* que não é garantida uma vez que é um decisão tomada pelo escalonador do sistema operacional, o qual a aplicação não tem controle.

O *bus* do ContextF permite ao componente que trata o evento decidir se ele deve utilizar a *thread* de execução principal da aplicação ou em uma outra *thread*. Porém, para ser aderente aos requisitos da plataforma, todos os componentes são executados em *threads* separadas providas pelo *thread pool* do *bus*.

Thread pool

É uma técnica presente no Java em que um conjunto de *threads* é previamente alocado pela aplicação para fazer melhor aproveitamento de *threads*. Funciona da seguinte forma: quando um processamento deve ser executado em paralelo, a aplicação requisita uma

thread ao *pool*. O *pool* por sua vez verifica se há alguma *thread* disponível e, caso houver, ela é utilizada para executar a requisição da aplicação. Quando o processamento termina, a *thread* é devolvida ao *pool* estando disponível para a aplicação novamente. Quando não há *threads* disponíveis, a requisição é colocada em uma fila que será consumida pelo *pool* assim que houver uma *thread* disponível. A Figura 5.2 mostra um exemplo do esquema de *thread pool*.

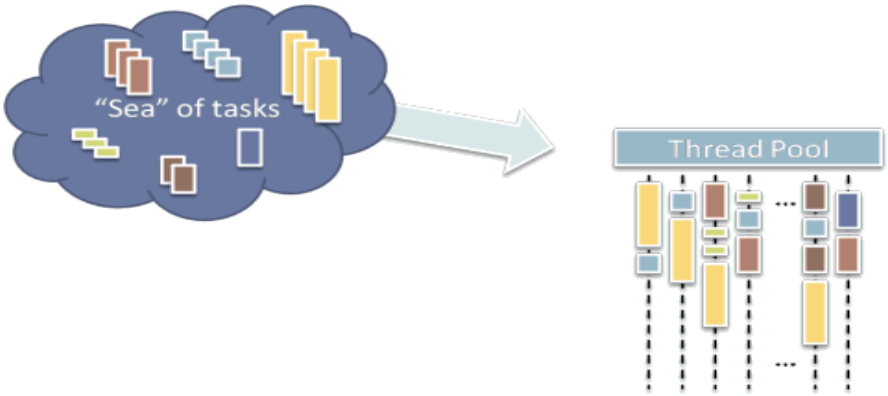

Figura 5.2: *Thread pool* Fonte: http://www.drdobbs.com/parallel/use-thread-pools-correctly-keep-tasks-sh/216500409

5.2 *ContextBarrier*

Barreiras comumente são utilizadas em situações onde existe um processo composto por diversas etapas e em algumas delas um conjunto de condições devem ser satisfeitas antes de prosseguir para a próxima etapa.

O mesmo pode ser utilizado para o caso de *threads* exigindo que, para um conjunto avance para a próxima etapa todas elas estejam no mesmo ponto, essa é a condição que deve ser satisfeita nesse caso. A Figura 5.3 mostra um esquema básico do funcionamento de barreiras.

ContextBarrier é o componente do *bus* responsável por implementar o mecanismo de *thread barrier* e garantir que a etapa de sincronização apenas comece quando todos componentes de extração de informações do contexto terminarem de executar.

Cada componente ao terminar de executar dispara um evento especial tratado apenas pelo *ContextBarrier* e como em todo evento uma nova *thread* é utilizada. Para

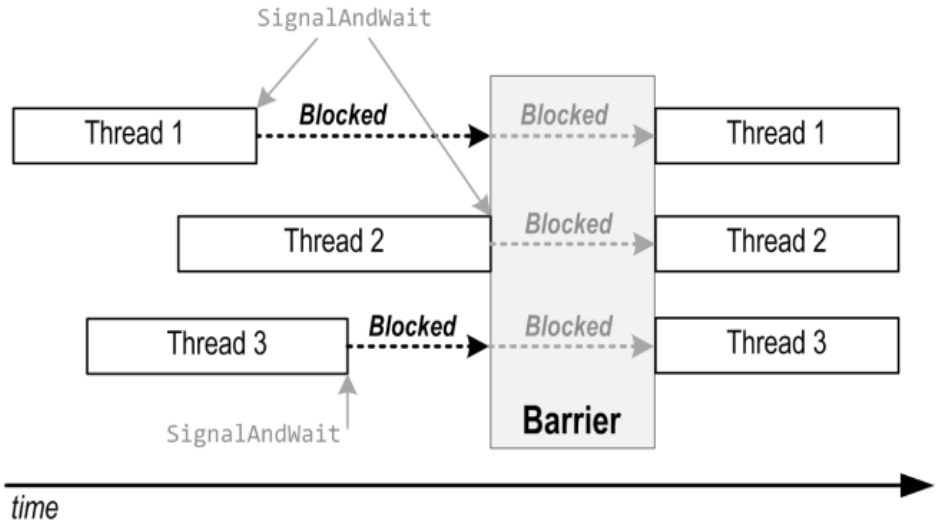

Figura 5.3: Barreira de *threads* Fonte: http://www.albahari.com/threading/part4.aspx

tanto, *ContextBarrier* utiliza o conceito de inteiros atômicos presentes no Java desde a versão 6. Esses inteiros possuem operações de incremento e decremento que são garantidas de serem executadas de forma atômica.

Então, quando o *ContextBarrier* recebe um evento, ele procura no registro a transação ao qual aquele evento faz parte. De posse dessa informação ele consegue verificar o número de componentes de extração de informações do contexto e assim quando o seu contador, que começa de 1, for igual ao número de componentes, o *ContextBarrier* dispara o evento de sincronização.

5.3 *TransactionRegistry*

Quando um evento que indica o inicio de um processo de contextualização de um recurso é disparado, ContextF gera uma nova transação para aquele processo. Essa transação é responsável por manter informações sobre cada processo em execução de forma isolada.

Cada evento que flui no *bus* tem a identificação de qual transação ele pertence permitindo que qualquer componente possa recuperar a transação associada a ele enquanto ela ainda não encerrou.

5.4 Extração de informações do contexto

A primeira etapa do processo de contextualização de um recurso é a extração das informações do contexto. ContextF por padrão contém quatro fontes de contexto e permite que a aplicação crie implementações customizadas de novas fontes de contexto.

5.4.1 *Context annotations*

Cada recurso capturado é composto por um tipo e um conjunto de anotações contextuais que representam cada tipo de informação que pode ser extraída do contexto conforme mostra o diagram 5.4. por dispositivos móveis.

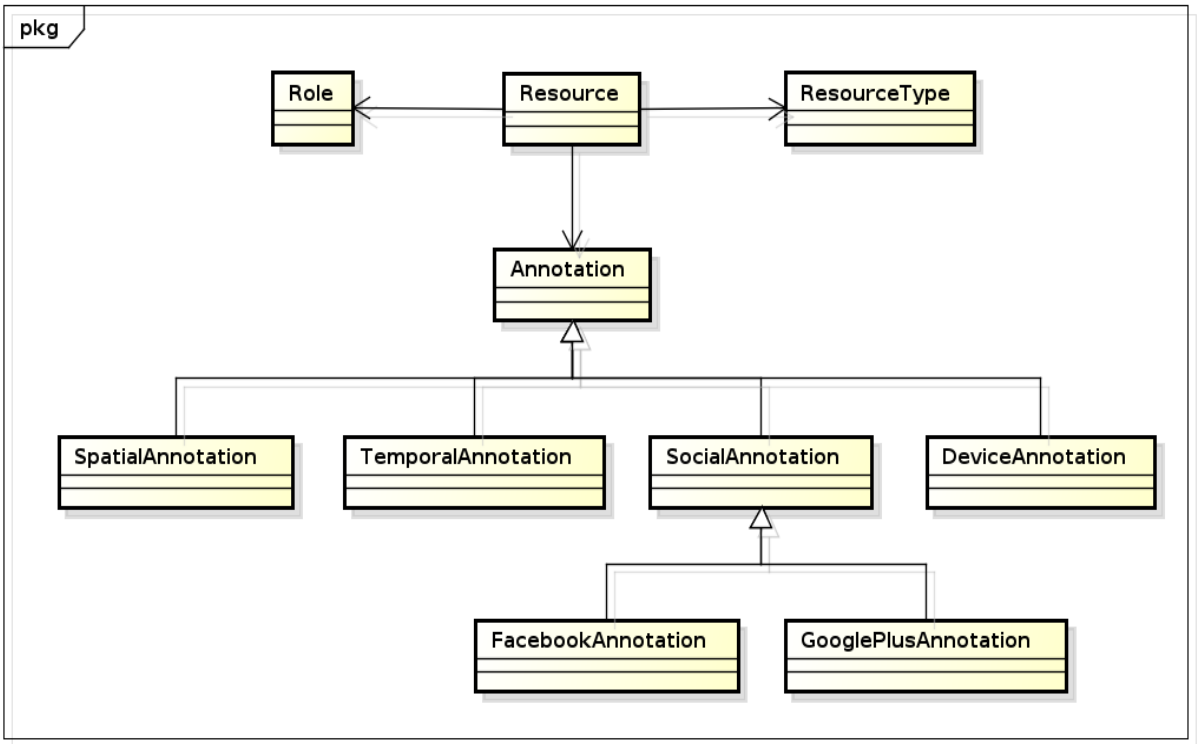

Figura 5.4: Recurso e suas anotações

- ***Resource***: representa um recurso capturado pela aplicação e contém a URL da localização do mesmo no *Media content server*.
- ***ResourceType***: representa o tipo do recurso.
- ***Role***: representa o papel do usuário no momento da captura do recurso.
- ***Annotation***: representação base para todas informações obtidas do contexto.

- ***SpatialAnnotation***: representa informações relacionadas a localização do recurso como latitude e longitude.

- ***TemporalAnnotation***: representa informações temporais como data e hora em que o recurso foi capturado bem como data e hora de quando foi sincronizado com o servidor de contexto.

- ***SocialAnnotation***: representação base de todas informações relacionadas a redes sociais.

- ***FacebookAnnotation***: representa uma especialização de anotação social contendo o *user-id* do Facebook.

- ***GooglePlusAnnotation***: representa uma especialização de anotação social contendo o *user-id* do Google Plus.

- ***DeviceAnnotation***: representa informações relacionadas ao dispositivo como resolução.

5.4.2 Componentes de extração

Constituem a etapa de contextualização do processo. Cada componente é responsável por lidar com um contexto específico. O diagrama de classes 5.5 mostra as fontes de contexto padrão do ContextF, bem como a hierarquia de classes que deve ser seguida por aplicações que implementarem componentes customizados.

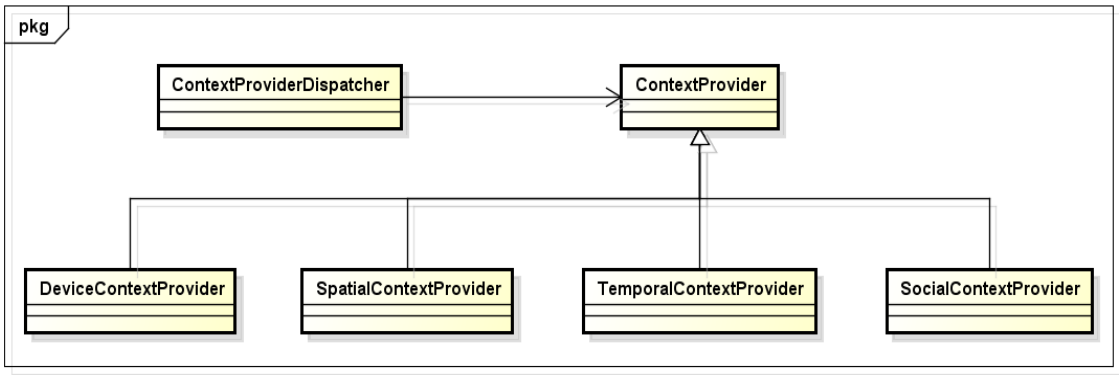

Figura 5.5: Provedores de contexto

- ***ContextProviderDispatcher***: é o principal componente responsável por invocar os provedores de contexto padrão e os definidos pela aplicação via arquivo de configuração do ContextF. Note que esse componente não possui uma referência direta para um provedor específico o que garante a ele maior desacoplamento e a possbilidade de invocar componentes padrão e customizados.

- ***ContextProvider***: interface que define os serviços que todo provedor de contexto deve implementar. Provedores customizados devem implementar essa interface também.

- ***DeviceContextProvider***: provedor de informações de contexto relacionadas ao dispositivo. Nesse provedor, por exemplo, é capturada a resolução do dispositivo que é importante no tratamento de resolução de imagens e vídeos.

- ***SpatialContextProvider***: provedor de informações de contexto relacionadas a sensores de localização. No Android há duas formas de obter a localização através da rede *wifi* e GPS. No caso, ContextF utiliza o GPS, pois gera uma localização mais precisa.

- ***TemporalContextProvider***: provedor de informações de contexto relacionadas ao relógio do dispositivo. Existem vários problemas associados a esse tipo de informação como fuso horários, porém, para os propósitos desta monografia será adotado que o relogio do dispositivo e do servidor estejam sincronizados.

- ***SocialContextProvider***: provedor de informações de contexto relacionadas a redes sociais. ContextF utiliza o próprio mecanismo de gerenciamento de contas do Android para obter o *user-id* associado ao dispositivo.

5.5 Sincronização com o servidor de contexto

É a segunda etapa do processo de contextualização, responsável por sincronizar as informações de contexto obtidas na primeira etapa. Essa etapa é implementada pelo componente de sincronização.

5.5.1 Componente de sincronização

Uma vez que todas informações foram obtidas, *ContextBarrier* dispara um evento que aciona esse componente. Ele é responsável por gerar o XML que será enviado ao servidor. O código a seguir mostra a estrutura desse XML.

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
    <media-content-url></media-content-url>
    <resource-type></resource-type>
    <role></role>
    <annotation>
        <temporal created-at="" synced-at=""/>
        <spatial latitude="" longitude=""/>
        <social user-id=""/>
        <device resolution=""/>
    </annotation>
</resource>
```

Note que essa é uma representação em formato XML do modelo de classes definido para a anotação de informações do contexto a recurso. Contudo, em momentos de falta de conexão com a Internet, ContextF deve agir de forma que esse processo de contextualização não seja perdido e a responsável por isso é a etapa de armazenamento.

5.6 Armazenamento

Etapa do processo responsável por prover armazenamento persistente de informações em situações de falta de conexão com a Internet.

5.6.1 Tipos de armazenamento

Existem algumas formas de prover esse armazenamento. No contexto de dispositivos móveis, duas soluções são viáveis.

- **Arquivo**: simplesmente escreve o XML gerado em um arquivo. Apesar de ser uma solução simples, gera uma série de problemas relacionados ao gerenciamento do acesso ao *hardware* de armazenamento primário e a realização de buscas.

- **SQLite**: é um SGBD relacional muito utilizado em sistemas embarcados. Apesar de ser baseado na utilização de arquivos ordinários que contém tabelas, índices, *triggers* e etc, seu formato é portável podendo ser transferidos entre máquinas arquiteturas de 32-bits e 64-bits, além de implementar quase todas funcionalidades do SQL. SQLite é o principal mecanismo de persistência da plataforma Android, e portanto, escolhida para a implementação do componente de armazenamento descrito a seguir.

5.6.2 Componente de armazenamento

Quando o componente de sincronização não pode enviar as informações para o servidor de contexto, um evento indicando essa impossibilidade é disparado.

ContextF, através desse componente, fornece um mecanismo de recuperação da falha na sincronização, para que esse processo de contextualização não seja perdido. Sua função é persistir essas informações para que elas possam ser sincronizadas com o servidor de contexto posteriormente.

Além do evento disparado pelo componente de sincronização, ele pode ser acionado por um evento indicando que deve ocorrer a sincronização de todas informações que ainda não foram enviadas ao servidor de contexto.

5.7 Monitoramento

ContextF possui uma etapa especial que não faz parte diretamente do processo de contextualização mas sim na manutenção do mesmo.

5.7.1 Componente de monitoramento

É responsável por monitorar o estado da rede para identificar quando há conexão com a Internet. Esse componente é implementado como um serviço do Android e executa em *background*.

Contudo, esse tipo de monitoramento deve ser executado com cautela devido ao consumo da bateria gerado por esse componente. Para tanto, ContextF permite que o usuário controle a ativação desse serviço de forma que faça o gerenciamento do tempo de vida de sua bateria de forma mais otimizada.

5.8 Configuração

Como o propósito desta monografia é a criação de *framework* genérico é necessário que cada aplicação possa configurar certos aspectos do mesmo.

5.8.1 Componente de configuração

ContextF suporta configurações no formato XML. Esse componente é responsável por transformar essas configurações do XML para o modelo de classes utilizado pelo ContextF. Para que o ContextF consiga localizar esse arquivo, ele deve estar presente em uma localização pré-definida pelo *framework*.

- **Definindo a localização do servidor de contexto**: permite que a aplicação informe ao ContextF a localização do servidor de contexto. Para tanto, existe o atributo *url*.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <context-server url="http://context.server.com"/>
    <context-provider>
        <temporal/>
        <spatial/>
        <social/>
        <device/>
    </context-provider>
</configuration>
```

- **Desabilitando um provedor de contexto**: permite que a aplicação escolha desabilitar um provedor de contexto. Para tanto, existe o atributo *enabled*.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <context-server url="http://context.server.com"/>
    <context-provider>
        <temporal enabled="false"/>
        <spatial/>
        <social/>
        <device/>
    </context-provider>
</configuration>
```

- **Criando um provedor de contexto customizado**: permite que as aplicações criem provedores de contexto customizados para a etapa de extração de informações do contexto. Dessa forma, outros repositórios de informações de contexto além dos fornecidos pelo ContextF, podem ser utilizados no processo de forma transparente para a aplicação.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <context-server url="http://context.server.com"/>
    <context-provider>
        <custom-provider>
            com.custom.provider.CalendarProvider
        </custom-provider>
        <temporal/>
        <spatial/>
        <social/>
        <device/>
    </context-provider>
</configuration>
```

6 Estudo de caso

O estudo de caso desta monografia é a construção de uma aplicação móvel para a plataforma Android, que utiliza o ContextF para a contextualização de imagens capturadas pelo dispositivo.

Tudo começa no momento em que o usuário acessa a aplicação, ele será autenticado via sua conta em redes sociais como Google e Facebook. Além disso, o usuário deve escolher que papel ele quer associar a imagem capturada, que o permite segregar suas imagens e definir regras de privacidade específicas para cada uma delas.

Quando capturada, a imagem deve ser enviada para algum serviço de armazenamento persistente como Instagram, Facebook ou qualquer outro escolhido pelo usuário através das preferências da aplicação.

Uma vez que o processo de envio terminou, a aplicação inicia o processo de contextualização do ContextF para a imagem capturada. As seguintes figuras exibem cada uma das telas da aplicação do estudo de caso.

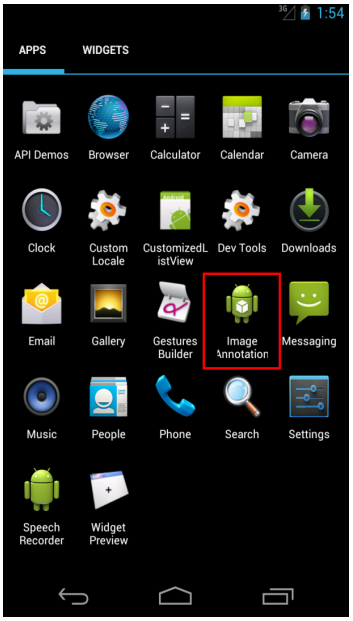

Figura 6.1: Menu de aplicações do Android

O destaque em vermelho na figura 6.1 mostra qual o ícone clicar para executar a aplicação.

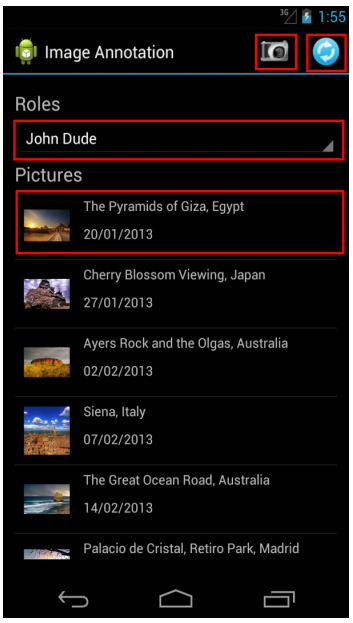

Figura 6.2: Tela de entrada da aplicação

A Figura 6.2 exibe a primeira tela da aplicação. Nela existem quatro informações relevantes: o ícone de ativação da câmera; o ícone de atualização com o servidor de contexto; a escolha do papel; as imagens capturadas juntamente a algumas informações básicas de contexto.

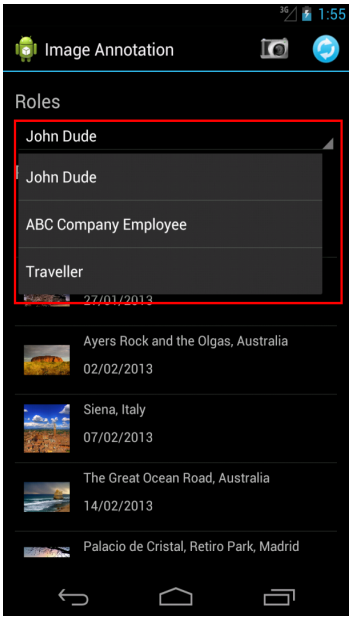

Figura 6.3: Tela de escolha do papel

A figura 6.3 mostra o componente de escolha dos papéis que são obtidos do servidor de contexto.

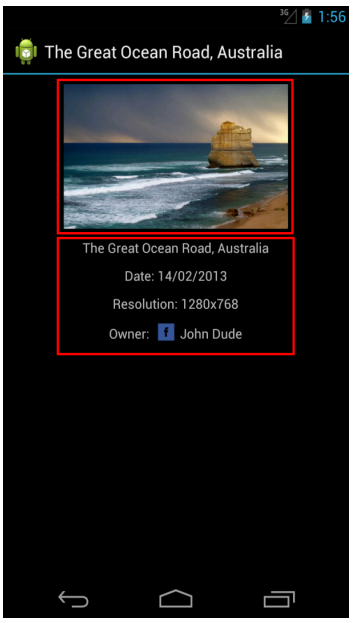

Figura 6.4: Tela que exibe os detalhes da imagem

A figura 6.4 mostra a tela com as informações de contexto detalhadas como: localização derivada a partir das coordenadas obtidas do sensor de GPS; data da captura; resolução do dispositivo que capturou a imagem; perfil de rede social de quem capturou a imagem.

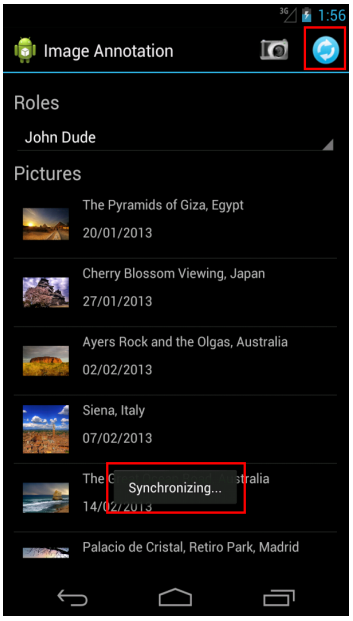

Figura 6.5: Atualizando com o servidor de contexto

A figura 6.5 mostra a tela quando o usuário clica no ícone de atualização. Note que essa atualização executa duas ações: obter novas imagens e informações de contexto

capturadas; disparar a etapa de sincronização do processo de contextualização do ContextF.

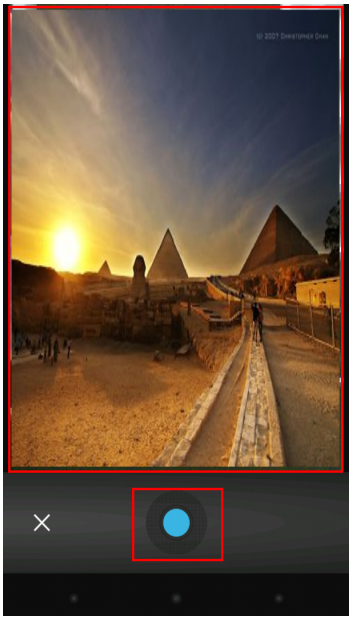

Figura 6.6: Acessando a camera do dispositivo via aplicação

Por fim, a figura 6.6 mostra a tela da câmera do dispositivo quando o usuário clica no ícone de ativação.

7 Conclusão

Devido a evolução dos dispositivos móveis, seus usuários tem, frequentemente, os utilizado para executar suas tarefas diárias. Bscando melhorar os serviços fornecidos aos seus usuários, contexto tem sido uma característica cada vez mais importante no desenvolvimento de aplicações móveis capazes de alterar seu comportamento a fim de fornecer serviços de forma adequada. Contudo, há toda uma lógica de interação com fontes de contexto e gerenciamento das informações obtidas que essas aplicações devem implementar e manter. Levando esses fatores em consideração, a criação de ferramentas que simplifiquem a interação com o contexto tornou-se um ponto chave na construção desse tipo de aplicação.

ContextF tem o desafio de ser um meio simples, transparente, desacoplado e customizável de interação com o contexto, que possa ser utilizado por essas aplicações. Para tanto, ele é baseado em uma arquitetura orientada a eventos que coordena toda comunicação entre os componentes que a compõem. Contudo, devido a necessidade de fazer uso otimizado dos recursos do dispositivo, uma série de problemas relacionados ao uso otimizado de recursos aparecem por esses dispositivos serem extremamente concorrentes. Uma das vantagens possibilitadas pelo uso de arquiteturas orientadas a eventos, é o comportamento reativo dos componentes que somente serão acionados quando houver algum processamento a ser realizado, otimizando o consumo de recursos. Outra vantagem é o desacoplamento permitido por esse tipo de arquitetura que possibilita um *framework* altamente customizável.

Embora ContextF seja um *framework* que cumpre o seu objetivo com funcionalidades simples e adaptáveis, ainda existem outras em aberto que podem ser abordadas em trabalhos futuros como o desenvolvimento de mecanismos mais robustos de autenticação e checagem de integridade e capacidade de derivar contextos complexos utilizando informações do próprio dispositvo móvel.

Referências Bibliográficas

Addlesee, M.; Curwen, R.; Hodges, S.; Newman, J.; Steggles, P.; Ward, A. ; Hopper, A. Implementing a sentient computing system. **Computer**, v.34, n.8, p. 50–56, 2001.

Bardram, J. E. **The java context awareness framework (jcaf)–a service infrastructure and programming framework for context-aware applications**. In: Pervasive Computing, p. 98–115. Springer, 2005.

BRAGA, R. B.; MARTIN, H.; VIANA, W. ; ANDRADE, R. M. C. **Captain: geraÃ§Ã£o de diÃ¡rios de bordo digitais usando anotaÃ§Ã£o contextual e conteÃºdo multimÃdia**. In: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web, p. 23, 2011.

Neto, R. D. F. B.; PIMENTEL, M. D. G. C. **Um processo de software e um modelo ontológico para apoio ao desenvolvimento de aplicações sensíveis a contexto**. 2006. Tese de Doutorado - Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação-ICMC-USP.

Carrizo, C.; Hatalkar, A.; Memmott, L. ; Wood, M. **Design of a context aware computing engine**. In: Intelligent Environments, 2008 IET 4th International Conference on, p. 1–4. IET, 2008.

Dey, A. K.; Abowd, G. D. ; Salber, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. **Human-computer interaction**, v.16, n.2, p. 97–166, 2001.

Hong, J. I.; Landay, J. A. **An architecture for privacy-sensitive ubiquitous computing**. In: Proceedings of the 2nd international conference on Mobile systems, applications, and services, p. 177–189. ACM, 2004.

Lo, D.; MendonÃ§a, P. R. ; Hopper, A. Trip: A low-cost vision-based location system for ubiquitous computing. **Personal and Ubiquitous Computing**, v.6, n.3, p. 206–219, 2002.

Korpipaa, P.; Mantyjarvi, J.; Kela, J.; Keranen, H. ; Malm, E. J. Managing context information in mobile devices. **Pervasive Computing, IEEE**, v.2, n.3, p. 42–51, 2003.

Kramer, R.; Modsching, M.; Schulze, J. ; ten Hagen, K. **Context-aware adaptation in a mobile tour guide**. In: Modeling and using context, p. 210–224. Springer, 2005.

Michelson, B. M. Event-driven architecture overview. **Patricia Seybold Group**, v.2, 2006.

Park, H.; Lee, J. **A framework of context-awareness for ubiquitous computing middlewares**. In: Computer and Information Science, 2005. Fourth Annual ACIS International Conference on, p. 369–374. IEEE, 2005.

Vieira, V.; Brézillon, P.; Salgado, A. C. ; Tedesco, P. A. A context-oriented model for domain-independent context management. **Revue d'intelligence artificielle**, v.22, n.5, p. 609–627, 2008.

Wang, X. H.; Zhang, D. Q.; Gu, T. ; Pung, H. K. **Ontology based context modeling and reasoning using owl**. In: Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, p. 18–22. IEEE, 2004.

Weiser, M. The computer for the twenty-first century. **Scientific American**, p. 94–101, 1991.

Weiser, M. **The computer for the 21st century, Human-computer interaction: toward the year 2000**. Morgan Kaufmann Publishers Inc., 1995.

Williamson, C.; Bernhard, J. T. ; Chamberlin, K. Perspectives on an internet-based synchronous distance learning experience. **Journal of Engineering Education**, v.89, n.1, p. 53–61, 2000.