Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Ciência da Computação

Uma Abordagem Baseada em Componentes para o Desenvolvimento de Aplicações Pervasivas Cientes de Contexto de Ambiente: Foco em Sensores

Bruno Fábio de Farias Paiva

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Hyggo Oliveira de Almeida
Angelo Perkusich

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

P149a Paiva, Bruno Fábio de Farias.
Uma abordagem baseada em componentes para o desenvolvimento de aplicações pervasivas cientes de contexto de ambiente: foco em sensores / Bruno Fábio de Farias Paiva. ¯ Campina Grande, 2016.
107 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) ¯ Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2016.
"Orientação: Prof. Dr. Hyggo Oliveira de Almeida; Coorientação: Prof. Dr. Angelo Perkusich".

1. Computação Pervasiva. 2. Computação Ubíqua. 3. Computação Móvel. 4. Desenvolvimento Baseado em Componentes. 5. Serviços Pervasivos. I. Almeida, Hyggo Oliveira de. II. Perkusich, Angelo. III. Título.

CDU 004.4 ¯2(043)

Resumo

A computação pervasiva é um paradigma em que o computador se torna onipresente e invisível para o usuário, com capacidade de obter informações acerca do ambiente ao redor e utilizá-las para controlar, configurar e ajustar aplicações dinamicamente. Os sistemas pervasivos se caracterizam pelo uso de sensores disponíveis no ambiente, cujos dados são processados para prover serviços personalizados para os usuários. Atualmente, o principal gerador de dados de sensores é o dispositivo portátil pessoal, como *smartphone* e *tablet*, pois são dispositivos que possuem diversos sensores embutidos. Do ponto de vista de desenvolvimento de *software*, tem-se um grande esforço na aquisição, tratamento e processamento dos dados de sensores, principalmente considerando diferentes plataformas, modelos de dispositivos e fabricantes. Neste trabalho, propõe-se uma abordagem baseada em componentes para o desenvolvimento de aplicações pervasivas baseadas em contexto de ambiente, ou seja, que utilizam sensores como base de dados para prover serviços para o usuário. A abordagem foi validada utilizando um experimento de desenvolvimento de aplicações para a plataforma Android. Os resultados indicam redução no tempo de desenvolvimento e no número de linhas de código geradas ao utilizar a abordagem proposta.

Palavras-Chave: Computação Pervasiva; Computação Ubíqua; Computação Móvel; Desenvolvimento Baseado em Componentes; Serviços Pervasivos.

Abstract

The pervasive computing is a paradigm which the computer becomes ubiquitous and invisible to the user with the ability to get information about the surrounding environment and use them to control, configure and tune applications dynamically. Pervasive systems are characterized by the use of sensors, which capture data from the environment, and then processes, to provide personalized services to users. In certain environments, the main sensor data generator is the personal portable device, which has several built-in sensors, such as smartphones and tablets. From a software development perspective there is a great effort in acquisition, treatment and data processing, especially considering different platforms, device model and device manufacturers. In this work, we propose a component-based approach to develop pervasive applications based on the environmental context by providing services which uses sensor data. The validation of our approach was an experiment which developers used the Android platform. Results show a reduction in the development time and the number of lines generated using this approach.

Agradecimentos

Agradeço primeiramente aos meus pais, meus irmãos e à minha namorada que sempre me apoiaram durante toda essa jornada de estudos, me incentivando, aconselhando, e me guiando nos momentos difíceis em que eu precisava de uma força. Em especial, agradeço-os pela compreensão e paciência.

Agradeço à Deus acima de tudo, pois sempre posso buscar forças nele não só nos momentos de fraqueza como nos de vitória também.

Também, aos meus orientadores, professor Hyggo Oliveira de Almeida e Angelo Perkusich. Sou muito grato pela paciência, compreensão e auxílio, pela dedicação empenhada, pelas motivações, pelos construtivos *feedbacks* e por todos os ensinamentos que me foram passados durante nossas aulas e reuniões de acompanhamento.

Aos integrantes do projeto de capacitação da Sony, por terem colaborado com meus experimentos do mestrado, em especial à Renata Derezzi, que colaborou bastante conseguindo o espaço físico e os alunos voluntários, possibilitando a execução do experimento sem maiores problemas.

A todos os meus amigos das turmas que participei durante o mestrado, com certeza foram muitos os aprendizados adquiridos durante as aulas.

Aos amigos mais próximos que durante toda a trajetória do mestrado deram força e conselhos tanto profissionais quanto pessoais além de me ajudarem no que fosse necessário.

Aos amigos do Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded), em especial ao frequentadores da sala 108, pela força e pelo bom humor no dia a dia.

Por fim, agradeço ao CNPq, pois o presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

Conteúdo

Lista de Siglas

CP - *Computação Pervasiva*

CpUb - *Computação Ubíqua*

CM - *Computação Móvel*

DBC - *Desenvolvimento Baseado em Componentes*

DC - *Desenvolvimento de Componentes*

API - *Application Programming Interface*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Nos últimos anos, com o avanço das tecnologias relacionadas aos dispositivos portáteis inteligentes, bem como da comunicação sem fio, o usuário pôde obter recursos computacionais e seus serviços em qualquer lugar e a qualquer momento. Este cenário deu origem a várias áreas especializadas da computação, tais como a computação pervasiva [18].

Mark Weiser descreve a computação pervasiva como componentes de *hardware* e *software* interligados que se tornam imperceptíveis ao usuário. Segundo o mesmo autor, os seres humanos estariam cercados pela computação e tecnologias de redes em seu ambiente. Além disso, os computadores forneceriam sistemas inteligentes que conectados entre si, estariam a trocar informações independentes, tornando-se então onipresentes [35].

A computação pervasiva pressupõe uma visão em que o computador está embarcado ao ambiente de forma invisível para o usuário, com capacidade de obter informações acerca do ambiente ao redor e utilizá-la para controlar, configurar e ajustar aplicações ao ambiente [7].

Segundo Merk L. *et.al.* [14], uma aplicação pervasiva deve apresentar as quatro seguintes características: suporte à mobilidade, leitura do contexto do usuário, adaptabilidade e descoberta de serviços disponíveis. A primeira característica sugere que independentemente de localização e do estado atual do dispositivo, a aplicação deve permanecer operando com conectividade e de acordo com o contexto corrente. O contexto traduz-se pelos dados relacionados à presença do usuário e suas interações em dado ambiente, tais como seus dados cinemáticos (ex: translações, rotações, velocidades, aceleração, etc), dados do ambiente (ex: clima, temperatura, umidade do ar) e dados georreferenciados (ex: latitude e longitude). A leitura de contexto se dá através dos dados que os sensores capturam no meio em que o

dispositivo está inserido. A adaptabilidade diz respeito à aplicação ser dinâmica aos novos dados coletados. E a descoberta de serviços indica que serviços disponíveis no ambiente, *a priori* desconhecidos do usuário, podem ser descobertos e acessados pela aplicação.

Este trabalho está inserido no contexto de aquisição e manipulação de dados de sensores para aplicações pervasivas. Mais especificamente, tem-se como foco prover suporte ao desenvolvimento de aplicações que utilizam tais sensores como fonte de dados para ciência de contexto.

1.1 Problemática

Segundo Neal Lathia *et. al.* [15], a implementação de ferramentas de coleta de dados de sensores permanece árdua e desafiadora: exige uma compreensão das diferentes interfaces de programação do sensor, bem como as questões de investigação relacionadas com a construção de sistemas de amostragem do sensor.

A maioria das plataformas de *middleware* atuais focadas em ciência de contexto geralmente enfatizam arquiteturas e mecanismos eficazes para o processamento de contexto e modelagem [22], mas não fornecem suporte a nível de desenvolvimento adequado.

Atualmente, o principal gerador de dados de sensores pessoais é o dispositivo portátil, como *smartphone* e *tablet*, pois são dispositivos que possuem diversos sensores embutidos. Do ponto de vista de desenvolvimento de *software*, tem-se um grande esforço na aquisição, tratamento e processamento dos dados de sensores, principalmente considerando diferentes plataformas, modelos de dispositivos e fabricantes.

A ausência de mecanismos de reutilização e adaptação de módulos de *software* faz com que a implementação de requisitos de manipulação de sensores tenha impacto direto na produtividade. Neste sentido, o problema abordado neste trabalho é: como promover aumento de produtividade através da reutilização de módulos de *software* para aquisição e manipulação de dados de sensores em aplicações pervasivas?

1.2 Objetivos

Neste trabalho, tem-se como objetivo a concepção de uma abordagem para o desenvolvimento de aplicações pervasivas baseadas em contexto de ambiente, ou seja, que utilizam sensores como base de dados para prover serviços para o usuário.

Para prover reúso e aumento de produtividade, assim como para esconder a complexidade de manipulação dos sensores do desenvolvimento de aplicação, utiliza-se o paradigma de Desenvolvimento Baseado em Componentes. Uma biblioteca de componentes é proposta para encapsular o comportamento dos sensores, facilitando o processo de desenvolvimento de aplicações.

A abordagem foi validada utilizando um experimento de desenvolvimento de aplicações para a plataforma *Android*. Os resultados indicam redução no tempo de desenvolvimento e no número de linhas de código geradas ao utilizar a abordagem proposta.

1.3 Contribuições

A principal contribuição do trabalho está na análise e discussão sobre o encapsulamento e criação de componentes de mecanismos de coleta e manipulação de dados de sensores. Com a biblioteca de componentes proposta, tem-se uma redução no esforço de desenvolvimento através do reúso, habilitando um aumento da produtividade.

A implementação da biblioteca para dispositivos *Android* demonstra a viabilidade técnica da abordagem. A biblioteca foi utilizada no desenvolvimento de aplicações, o que respalda ainda mais a utilidade do resultado obtido.

Por fim, o experimento realizado permite identificar pontos de melhoria para pesquisas futuras. Este resultado serve como base para investimentos na área, alinhados com os objetivos do Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande.

1.4 Estrutura da Dissertação

O restante do documento está estruturado da seguinte forma:

- no Capítulo 2 são apresentados conceitos fundamentais para o entendimento do trabalho;
- no Capítulo 3 são discutidos trabalhos relacionados;
- no Capítulo 4 apresenta-se a abordagem proposta, incluindo a biblioteca de componentes proposta e o desenvolvimento de aplicações usando a biblioteca;
- no Capítulo 5 discute-se a validação da abordagem proposta;
- No Capítulo 6 apresentam-se as conclusões e as limitações do trabalho desenvolvido, bem como sugestões de pesquisas a serem exploradas futuramente.

Capítulo 2

Fundamentação Teórica

Neste capítulo, serão apresentados conceitos fundamentais para a compreensão do trabalho, tais como: Desenvolvimento Baseado em Componentes, Computação Móvel e Pervasiva. Neste sentido, serão detalhadas e apontadas as suas principais características, possíveis aplicações de uso, requisitos, dentre outros aspectos.

2.1 Computação Móvel

Com o avanço tecnológico e de pesquisas no campo da telecomunicação e redes de computadores, a nova tecnologia de comunicação de dados sem fio pôde se desenvolver. Tal abordagem de comunicação ocorre com auxílio de equipamentos de radiofrequência (comunicações via ondas de rádio), infravermelho, dentre outras [34]. Neste sentido, uma rede sem fio é uma infraestrutura das comunicações sem fio que permite a transmissão de dados e informações sem a necessidade do uso de cabos (ex: telefônicos, coaxiais ou óticos). A fim de regularizar o uso de tal tecnologia, diferentes padrões foram definidos pela comunidade científica e industrial, a fim de estabelecer protocolos específicos para diferentes tipos de aplicações (ex: de pequeno e longo alcance), sendo alguns deles já conhecidos: *bluetooth*, *wifi* e *wiMax*, dentre outros [34]. Neste contexto, novos paradigmas de computação puderam se desenvolver, com base nas novas infra-estruturas de rede disponíveis, como por exemplo a Computação Móvel. De acordo com Forman (1994), a Computação Móvel utiliza computadores e/ou dispositivos portáteis e comunicação de dados sem fio (*wireless*) com o intuito de permitir aos usuários trabalharem fora de ambientes fixos. O ambiente móvel baseia-se na

capacidade que os usuários têm, munidos de um dispositivo móvel (*laptops*, *handhelds*, etc), de se comunicarem com estações fixas da infra-estrutura de rede (ex: antenas, servidores) e possivelmente com outros dispositivos móveis [13]. Neste sentido, tal paradigma possibilita o aumento da capacidade de mover fisicamente serviços computacionais junto ao usuário, tornando os dispositivos móveis sempre presentes a fim de expandir a capacidade do mesmo utilizar estes serviços que o computador oferece, independentemente de sua localização. Diversas aplicações móveis têm surgido nas últimas décadas, tais como *softwares* colaborativos, sistemas de navegação e geolocalização, navegação para orientar os usuários em locais desconhecidos (ex: museus), dentre outras. Uma importante limitação da computação móvel é que o modelo computacional não muda enquanto nos movemos (não é adaptativo), isto é, o dispositivo não é capaz de obter flexivelmente dados sobre o contexto no qual a computação ocorre e ajustá-la corretamente, problema este solucionado pelas abordagens apresentadas a seguir. [30]

2.2 Computação Pervasiva

A rápida evolução e disseminação da tecnologia nos dias atuais vêm fazendo com que as pessoas incorporem no seu dia-a-dia o uso de computadores e dispositivos móveis, bem como impulsionando as aplicações no contexto da computação pervasiva [35]. A Computação Pervasiva é um conceito que define que os meios de computação estão inseridos no ambiente dos usuários de forma perceptível ou imperceptível, de modo que o computador ou dispositivo esteja distribuído no meio e não esteja apenas em cima de uma mesa, por exemplo [31]. Segundo Mark Weiser [35], a Computação Pervasiva obtém dados e informações sobre o contexto atual do usuário, através do uso de vários dispositivos, móveis ou dispositivos também conectados fisicamente a uma rede. Os sistemas computacionais pervasivos utilizam os sensores presentes nos dispositivos e a troca de informação através da rede em que os dispositivos estão conectados. Assim, o dispositivo móvel passa a ser um 'coletor' dinâmico das informações acerca do usuário. Dotados de sensores, os dispositivos têm a capacidade de detectar e extrair dados e as variações do ambiente, gerando automaticamente modelos computacionais, controlando, configurando e ajustando aplicações conforme as necessidades do usuário e dos demais dispositivos [7].

Com base neste novo modelo computacional, surgem novas aplicações inteligentes e que integram-se de forma transparente à vida das pessoas, auxiliando na execução de suas tarefas. Dentre várias funcionalidades, uma aplicação pervasiva deve fornecer recursos (informações e serviços) onde o computador possa oferecer serviços ao usuário, independentemente de local e hora [31]. Os sistemas pervasivos abrangem inúmeras áreas, dentre elas estão: *healthcare*, anúncios [24], [10] e jogos [4], [19]. No cenário de *healthcare*, por exemplo, a computação pervasiva tem contribuído bastante, devido ao acompanhamento eficaz que os dispositivos vestíveis (*wearable devices*) possibilitaram, trazendo consigo suporte para elaborar análises mais aprofundadas, uma vez que oferecem precisão nas medições realizadas [1], [8].

Adicionalmente, as informações de contexto deverão estar disponíveis às aplicações para que estas possam tomar decisões de forma inteligente e transparente aos usuários. Tais sistemas, possuem um alto grau de heterogeneidade devido à diversidade de recursos inseridos na rede e à dinamicidade de entrada e saída de usuários no ambiente [7].

Desta forma, é de real valia proporcionar uma infraestrutura de *hardware* e *software* que possibilite a abstração das complexidades de baixo nível presente em aplicações de computação pervasiva. Tais complexidades se relacionam ao processamento dos sinais de sensores obtidos, aspectos de conectividade entre dispositivos com diferentes interfaces de comunicação, a coleta das informações oriundas do ambiente e dos usuários (Ciência de Contexto), bem como os requisitos de segurança e controle de acesso aos recursos disponíveis na rede [17].

Como observa-se, é de fundamental importância esta etapa de processamento adequado dos sinais de baixo nível, uma vez que tais dados serão utilizados na representação das informações acerca das atividades e circunstância atual do usuário. Neste sentido, novos requisitos devem ser considerados pela metodologia de engenharia de *software* utilizada, tais como: ciência de contexto do usuário e a interoperabilidade. O último, surge como importante fator para que as informações de contexto possam ser compartilhadas entre diferentes dispositivos e plataformas, nos ambientes distribuídos [2].

Loureiro et. al., propõem as condições atuais do usuário, do ambiente no qual o mesmo se encontra e do próprio dispositivo computacional utilizado, considerando tanto suas características de *hardware*, como também de *software* e de comunicação [17]. Tais entradas são

os chamados contextos e diferentes definições têm sido propostas na comunidade acadêmica. Vários pesquisadores publicaram trabalhos no início da década de 90, descrevendo definições para o significado do termo contexto. Dey et. al. formalizaram a definição de contexto como sendo "qualquer informação que possa ser utilizada para caracterizar a situação de entidades (pessoa, lugar ou objeto) que sejam consideradas relevantes para interação entre um usuário e uma aplicação (incluindo o usuário e a aplicação)" [9].

O contexto pode ser compreendido como a junção de uma gama de variáveis que estão associadas ao estado corrente em que o usuário está inserido, tais como: iluminação do meio, nível de ruído, conectividade, largura de banda, localização, custos de comunicação [27].

Uma aplicação consciente de contexto é caracterizada pelos fatores relacionados à coleta do contexto através de sensores, bem como a maneira como o contexto é interpretado e a capacidade de utilizá-lo para realizar a execução da tarefa. Dessa forma, uma aplicação consciente de contexto se baseia principalmente em sensores de *hardware* e algoritmos de processamento de sinais [31]. Tal requisito é fundamental para que os ambientes de computação pervasiva estabeleçam uma relação entre as entidades computacionais envolvidas, com as capacidades técnicas e as demandas de cada usuário.

2.3 Desenvolvimento Baseado em Componentes

Componentes de *software* reusáveis são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e têm interfaces claras, além de documentação apropriada e uma condição de reúso definida. Um componente deve possuir as seguintes características para que sua implementação traga benefícios ao sistema, são elas:

1. Autocontido: característica de reúso, na qual os componentes são independentes de outros componentes. Caso exista alguma dependência, então todo o conjunto deve ser visto como o componente reutilizável;

2. Identificação: os componentes precisam ser distinguíveis de forma clara, para manter a facilidade em sua identificação, possibilitando uma fácil manutenibilidade do código através de simples operações de troca de componente.

3. Funcionalidade: componentes têm uma funcionalidade clara e específica que realizam e/ou descrevem. Componentes podem realizar funções ou podem ser simplesmente descrições de funcionalidades;

4. Interface: componentes devem ter uma interface clara, que indica como podem ser reusados e conectados a outros componentes. Esta interface funciona como o contrato entre as partes do sistema.

O Desenvolvimento Baseado em componentes (DBC) é um processo que enfatiza o projeto e a construção de sistemas baseados em componentes de *software* reutilizáveis [23]. A engenharia de componentes é uma derivação da engenharia de *software*, focada na decomposição dos sistemas, em componentes funcionais e lógicos com interfaces de utilização bem definidas, usadas para comunicação entre os próprios componentes, que estão em um nível mais elevado de abstração do que os objetos.

O DBC surge como uma nova perspectiva de desenvolvimento de *software* caracterizada pela composição de partes já existentes. Construir novas soluções pela combinação de componentes desenvolvidos e comprados aumenta a qualidade e dá suporte ao rápido desenvolvimento, levando à diminuição do tempo de entrega do produto ao mercado [33]. Os sistemas definidos através da composição de componentes permitem que sejam adicionadas, removidas e substituídas partes do sistema sem a necessidade de sua completa substituição. Com isso, DBC auxilia na manutenção dos sistemas de *software*, por permitir que os sistemas sejam atualizados através da integração de novos componentes e/ou atualização dos componentes já existentes.

Esta abordagem busca, através da composição de partes já existentes de *software*, trazer uma nova perspectiva de desenvolvimento. Construindo novas soluções através do encaixe de componentes bem definidos que obedecem a uma restrita interface de uso, torna-se possível obter maior qualidade em menos tempo de desenvolvimento. Os sistemas definidos através da composição de componentes permitem que partes do sistema sejam editadas sem a necessidade de sua completa substituição. Neste sentido, DBC traz avanços significantes na manutenção dos sistemas de *software*, por permitir atualizações através da integração de novos componentes [33].

As abordagens de DBC geralmente se assemelham a ilustrada na Figura 2.1, cujos elementos são descritos a seguir:

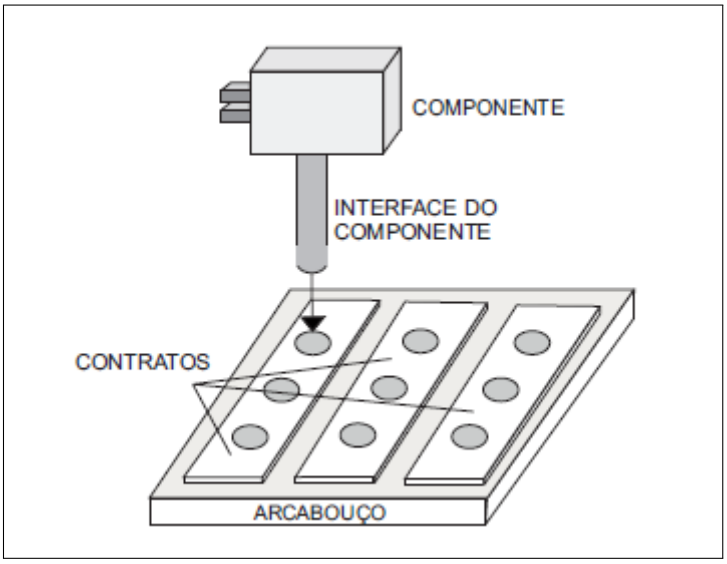


Figura 2.1: Representação da abordagem baseada em DBC por *Ferreira et. al* [12].

- Interface do Componente: representada pelas assinaturas dos métodos que implementam a funcionalidade do componente.

- Contratos: representa a descrição de interface, ou seja, a especificação do comportamento de um componente. O contrato é composto por três etapas: os Invariantes, que são as restrições que o componente deverá manter; as Pré-condições(parâmetros de entrada), que são as restrições que precisam ser satisfeitas pelo cliente; e as Pós-condições (parâmetros de saída) que são os resultados dos processamentos que o componente realizou com base nos dados obtidos como parâmetros de entrada, ou seja, são os dados que o componente irá retornar. Os contratos garantem que o desenvolvimento de componentes independentes obedeça determinados padrões, fazendo com que o arcabouço de componentes seja capaz de utilizá-los sem conhecer detalhes internos de implementação [3].

- Arcabouço de Componentes: é a infraestrutura que possibilita a "montagem" dos componentes para a construção de aplicações, coordenando a interação entre os componentes (*frameworks*, *middlewares*, etc). Para garantir que os componentes se comportarão da maneira esperada pelo arcabouço, são definidos os contratos.

Capítulo 3

Trabalhos Relacionados

Neste capítulo apresentam-se os principais trabalhos relacionados identificados através de revisão sistemática. Segundo Sampaio [26], uma revisão sistemática é uma forma de pesquisa que utiliza como fonte de dados a literatura acadêmica sobre determinado tema. Deste modo, é disponibilizado por tal recurso, um resumo das evidências relacionadas a um escopo de pesquisa, por meio da aplicação de métodos explícitos de busca, apreciação crítica e síntese da informação selecionada, sendo particularmente útil para integrar informações de um conjunto de diferentes tipos de estudos realizados. Segundo Magno [5], geralmente, tal procedimento envolve as seguintes etapas: (a) definição de pergunta(s) científica(s), (b) escolha de bases de dados, (c) definição das palavras-chave e estratégias de busca, (d) estabelecimento de critérios para a seleção (filtros), (e) conduzir busca, (f) aplicação dos filtros definidos, (g) análise crítica e avaliação final. No que diz respeito às questões de pesquisa, estas servem de norteamento e tem relação determinante para os critérios de busca, e consequentemente com os achados da pesquisa. Deste modo, foram levantados os seguintes questionamentos:

1. Como encontra-se o cenário atual da computação pervasiva na literatura mundial?
2. Quais mecanismos utilizados para captura de contexto dos usuários?
3. Quais funcionalidades e áreas de aplicação são oferecidas pelos sistemas de computação pervasiva?

O levantamento dos trabalhos foi realizado com o auxílio do portal de busca de periódicos da CAPES e da ferramenta de busca Google Acadêmico, com buscas feitas em vá-

rios servidores de publicações científicas, tais como: IEEE Xplore, Elsevier, SpringerLink, ACM Digital Library, CiteSeerX. As palavras-chave utilizadas nas buscas foram: Computação Pervasiva, Computação Ubíqua, Computação Móvel, Ciência de contexto, Sistemas ubíquos, Desenvolvimento Baseado em Componentes (DBC), Componentes. Os mesmos termos foram consultados em inglês: *Pervasive Computing*, *Ubiquotous Computing*, *Mobile Computing*, *Context awareness*, *Ubiquitous systems*, *Component Based Development*. Como chaves de consulta, tais termos foram combinados entre si. Os trabalhos correlatos encontrados foram filtrados de acordo com o nível de similaridade com o problema ou solução abordados neste trabalho.

3.1 Desenvolvimento Dinâmico de Ambientes Multimídia Cientes de Contexto

O trabalho de Duarte et. al. [11] emprega o DBC em sua arquitetura. Utilizando o *middleware* LoCCAM [20] já disponível na literatura, tem como foco evoluir o LoCCAM incrementando a funcionalidade de automatização do processo de instalação de componentes em aplicações cientes de contexto. Como pode ser visto em sua arquitetura, Figura 3.1, o trabalho se limita apenas ao sistema operacional *Android* devido ao próprio *middleware* ter essa restrição interna, porém a arquitetura sugerida aborda o problema de reconhecer o contexto do usuário de maneira dinâmica, ou seja, de acordo com os dados de sensores capturados ocorre a adaptação, em tempo de execução e também em tempo de desenvolvimento.

O uso de componentes se deu de forma semelhante a deste trabalho proposto, abordando também o encapsulamento dos sensores facilitando o reúso. Três tipos de componentes de aquisição de contexto foram definidos neste trabalho, são eles: físico, no qual engloba apenas o sensor propriamente dito; lógico, onde são feitas as inferências combinando os dados obtidos por diversos sensores; virtual, representado por um serviço *web* que fornece informações as quais apenas os sensores do dispositivo local não conseguiria, por exemplo. Com o intuito de facilitar a representação de Contextos, os autores utilizam um conceito modelado de forma hierárquica, denominado *Context Key* (ex: context.ambient.temperature). Em relação a nossa proposta (detalhada no Capítulo 4), apresentamos o conceito de Domínio (implementado pelo Componente de Domínio) que tem por objetivo facilitar a representação

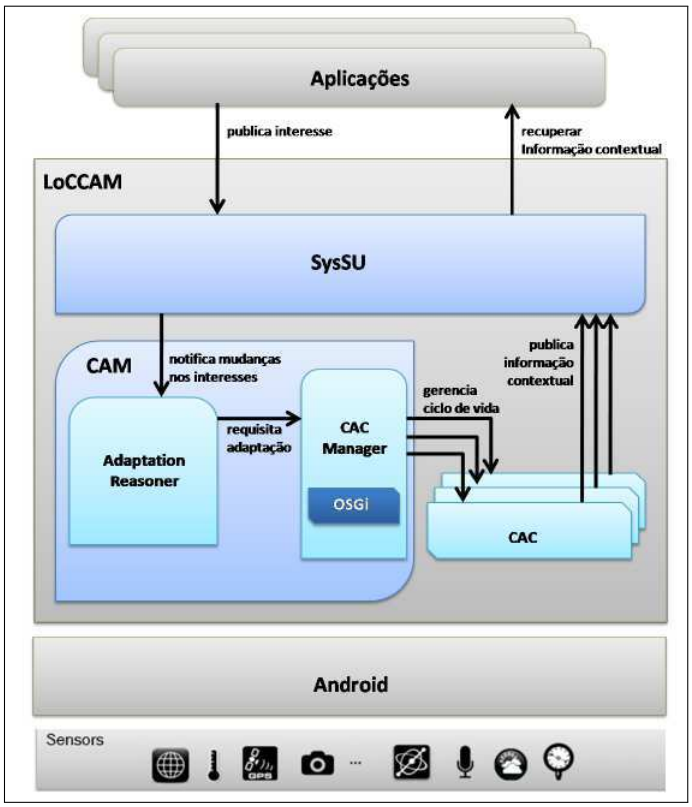


Figura 3.1: Representação da arquitetura do *middleware* LoCCAM por *Duarte et al*.

de contextos, mediante o agrupamento de sensores.

3.2 Um *Framework* Voltado para Prototipação Rápida com Conceitos de Interação Multi-Modal

Ronny Seiger et. al. [28], apresentam o *framework* chamado *Connect*, o qual é voltado para apoio à prototipagem de sistemas ubíquos, com enfoque em aspectos de engenharia de *software*. O objetivo da ferramenta é aprimorar a criação de protótipos de conceitos de interação. O *framework* permite a definição e modificação de mapeamentos de eventos e ações para dispositivos de interação e aplicações. Ao utilizar o *Connect*, protótipos baseados em modelos de conceitos de interação multimodais que envolvem vários dispositivos podem ser criados, avaliados e refinados durante todo o processo de engenharia.

Os componentes de sensores representam os dispositivos e aplicativos que são capazes de detectar interações e produzir eventos de interação correspondentes. Tais componentes

possuem em comum uma interface (*SensorComponent*), sendo necessário o uso de um adaptador para conectar o sensor através de sua *API* nativa ao *framework*. Outros componentes do sistema são os *ActuatorComponents* responsáveis pelo processamento dos eventos de interação, além dos *ComplexComponents*, que são encarregados de interconectar os *SensorComponents* aos *ActuatorComponents*. No entanto, é apontado pelos autores que os modelos de componentes não são capazes de processar valores com aspectos de incerteza em sua mensuração. Além disto, o uso de uma linguagem formal definida para descrever as interfaces de um sensor pode aprimorar a capacidade para adicioná-los em tempo de execução.

3.3 Um *Framework* para o Sensoriamento de Dados para Computação Pervasiva. *Mobile Autonomous Sensing Unit (MASU)*

O sistema proposto por Medina et.al [21] visa atender às restrições de dispositivos e ambientes operacionais que os tornam impróprios para acompanhamento de trabalhos fracamente acoplados ou totalmente distribuídos. Assim, o trabalho apresenta uma estrutura que suporta dados distribuídos invasivos de detecção de uma forma genérica. Segundo os autores deste trabalho, os desenvolvedores podem usar essa estrutura para facilitar as implementações de suas aplicações, reduzindo a complexidade e esforço em tal atividade.

O *framework* foi avaliado por meio de simulações e também através de um experimento, onde os resultados obtidos indicaram sua utilidade para apoiar essa atividade de detecção em cenários de trabalho fracamente acoplados ou totalmente distribuídos. O sistema é capaz de coletar e compartilhar dados entre dispositivos (*smartphones*) envolvidos nos ambientes distribuídos.

Neste trabalho não foram avaliadas medidas objetivas (diretas), apenas as de qualidade do produto final. O *framework* MASU foi avaliado para determinar os seus custos de uso e desempenho em termos de energia, computação e tráfego de rede. A avaliação incluiu simulações e também um teste empírico.

3.4 Um *Framework Web* para Aplicações Móveis Cientes de Contexto

No trabalho de Jianchao Luo e Hao Feng [16], é apresentado um *framework web*, chamado CamWAF, destinado a apoiar o rápido desenvolvimento de aplicações móveis sensíveis ao contexto e simplificar o intercâmbio de informações de contexto em ambientes heterogêneos. Neste sentido, observou-se que o sistema provê suporte as plataformas *Windows*, *iOS*, *Android*. CamWAF, assim como neste trabalho proposto, traz como objetivo diminuir o esforço necessário para produção e simplificar a tarefa de desenvolvimento de aplicativos sensíveis ao contexto.

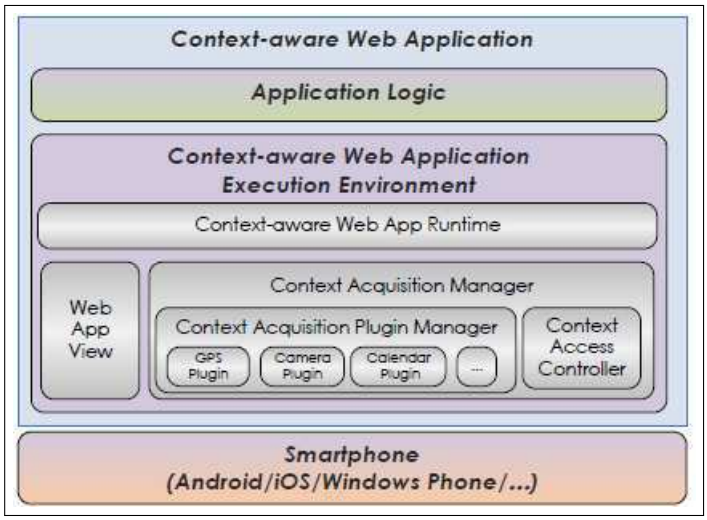


Figura 3.2: Representação da arquitetura *Web* apresentada por *Jianchao Luo e Hao Feng*.

A abordagem considera a limitação de recursos nos dispositivos móveis e com isso o *framework* delega os processamentos pesados para servidores *web*, enquanto permitem que as aplicações móveis sensíveis ao contexto possam consultar e salvar informações de contexto de alto nível de maneira mais leve para o dispositivo.

Este trabalho possui entidades semelhantes a alguns componentes propostos neste trabalho, porém com algumas particularidades. Por exemplo: *Context Aquisition Manager* seria uma entidade equivalente ao nosso componente de Gerenciamento, com a ressalva de que o foco da entidade é em aplicações *Web* e o do nosso componente é gerenciar os dados locais capturados pelos componentes de sensores.

3.5 Análise dos trabalhos

Os trabalhos relacionados apresentam variados tipos de estudos, envolvendo o desenvolvimento de aplicações pervasivas, discussões sobre análise de requisitos, arquiteturas, dentre outros estudos. De modo geral, as aplicações encontradas se destinam ao provimento de soluções, com base em aspectos de engenharia de *software*, a fim de facilitar a modelagem e reconhecimento de contexto. Os sistemas variam quanto aos recursos suportados, tais como: paradigma computacional(Pervasivo, Móvel), tipo de solução (*middleware*, *framework*, biblioteca e aplicação), tecnologias utilizadas, área de aplicação e principais requisitos atendidos (distribuição, escalabilidade, adaptabilidade e segurança).

De modo geral, observou-se que os sistemas de apoio ao desenvolvimento de aplicações cientes de contexto dos usuário utilizam-se de variados tipos de tecnologias de *hardware* (dispositivos) e *software* (*middlewares*, sistema operacional), com maior utilização da plataforma *Android*. No que tange aos requisitos de tais sistemas, basicamente os mesmos requisitos são minimamente atendidos de modo a obedecer o paradigma computacional correspondente ao foco de cada sistema (ex: escalabilidade, persistência, interação dos componentes), sendo os requisitos de Adaptabilidade, Interoperabilidade e Distribuição os mais atendidos. No paradigma da computação pervasiva, que é o foco deste trabalho, o uso de *middlewares* foi o mais abordado.

A Figura 3.3 apresenta uma visão geral sobre cada trabalho correlato mostrando qual o foco do trabalho, solução apresentada e limitação ou trabalho futuro. Como observado, o trabalho de Duarte et. al. (Seção 3.1) apresenta como limitação o fato de não prover independência de plataforma, uma vez que a sua solução (o *middleware* LoCCAM) é fortemente acoplada à plataforma *Android*, embora tenha sido testado e validado em versões diferentes da mesma. Em relação ao trabalho desenvolvido por Ronny Seiger et. al. (Seção 3.2), destacamos que a mesma se restringe à produção de protótipos de sistemas ubíquos interativos multimodais, sem prover uma abstração de *software* para geração de produtos finais com suporte ao tratamento dos dados de sensores. Outro aspecto limitante do trabalho é a ausência de experimento e testes de validação. No trabalho de Medina (Seção 3.3) a utilização da computação distribuída foi destacada como ponto positivo, porém não houve uma avaliação no quesito da produtividade alcançada em se tratando do uso de sua abordagem. O trabalho

de Luo J. e Feng (Seção 3.4) não fez uso da componentização em sua infraestrutura, perdendo as vantagens do reúso e da abstração das complexidades no desenvolvimento desses sistemas.

Trabalho	Foco	Solução	Limitação
Duarte et. al. (2015)	DBC (automatização)	Middleware (LoCCAM)	Interoperabilidade
Seiger et.al. (2015)	DBC (Prototipagem)	Framework (Connect)	Sem avaliação
Medina et. al. (2016)	Sensoriamento distribuído de dados	Framework (MASU)	Produtividade não avaliada
Jianchao Lu, Hao Feng (2016)	Desenvolvimento de aplicações pervasivas	Framework (CamWAF)	Não faz uso de Componentes

Figura 3.3: Visão geral dos trabalhos relacionados.

Uma das diferenças entre a estratégia apresentada neste trabalho e as citadas é que a infraestrutura fornecida também oferece suporte a cenários pré-definidos. Cenários estes nos quais diferentes contextos podem ser reconhecidos com maior facilidade, além de manter seu foco no desenvolvimento baseado em componentes, suprindo as demandas particulares e abstraindo aspectos de implementação ao usuário final (os desenvolvedores). Foi produzido também neste trabalho a biblioteca *PervasiveSensorLib*, que é um exemplo de instância da abordagem de componentes sugerida aplicada à plataforma *Android*, que utiliza a componentização dos sensores.

Capítulo 4

Abordagem Proposta

Neste capítulo é apresentado, detalhadamente, a abordagem utilizada para a especificação e o desenvolvimento da biblioteca *PervasiveSensorLib*, além de exemplificar seu uso através de implementações baseadas nos componentes criados.

4.1 *PervasiveSensorLib*: Uma biblioteca de componentes

A biblioteca *PervasiveSensorLib* foi desenvolvida com o objetivo de simplificar o desenvolvimento e manutenção de aplicações móveis adaptáveis, sensíveis ao contexto. Esta biblioteca não só apoia a composição, a reutilização e o gerenciamento do uso dos sensores, mas também fornece uma abstração de nível de programação que facilita a codificação. Esta facilidade se dá através do uso dos componentes nela disponíveis, além da definição de perfis para utilização de um conjunto de sensores direcionados ao mesmo propósito. Neste contexto, a criação de aplicações pervasivas, que utilizam os sensores de dispositivos móveis inteligentes, podem capturar as características do meio em que o usuário final se encontra através dos componentes concebidos na abordagem e que são especificados neste capítulo.

De modo geral, a Figura 4.1 mostra uma representação da arquitetura da abordagem proposta, contendo as camadas de Aplicação, de *Middleware*, de Sistema Operacional (S.O.) A camada física é responsável por fazer a captura dos dados brutos dos sensores presentes no dispositivo e transferí-los para a camada do S.O. a qual apresenta como função, implementar a *API* nativa do S.O. específico para acessar os dados de sensores coletados pelo dispositivo, visando a interoperabilidade. A camada de *Middleware*, onde se encontra a biblioteca de

componentes, define os seguintes tipos de componentes: Componentes de Sensores; Componentes de Gerenciamento; Componentes de Interação e os Componentes de Domínio. O papel da camada de *Middleware* é fornecer para a camada da aplicação todos os dados de sensores já processados de acordo com os requisitos da aplicação previamente definidos.

Figura 4.1: Representação da arquitetura da abordagem proposta.

A seguir serão detalhados cada um dos componentes presentes na abordagem, consequentemente, presentes na biblioteca *PervasiveSensorLib*.

4.1.1 Componentes de Sensores

Os principais componentes pertencentes à biblioteca são os que encapsulam as funcionalidades dos sensores. O modelo definido para especificação destes componentes é o seguinte: sensor utilizado; tipo/perfil do sensor; parâmetros de entrada; e parâmetros de saída.

Os Componentes de Sensores tratam os dados recebidos por um, e somente um, tipo de sensor. Eles recebem os dados brutos de um "Componente de Comunicação"o qual tratam os dados relevantes e os enviam para um "Componente de Ponte"cuja função se dá pelo fornecimento de dados para o cliente através de uma interface, ambos componentes serão

abordados na seção 4.1.3 deste capítulo. Os componentes de sensores são descritos nas subseções seguintes.

Componente Temperatura Ambiente *(Ambient Temperature)*

O componente temperatura encapsula a funcionalidade do sensor de temperatura, e tem como principal objetivo fornecer ao cliente a temperatura apresentada no meio em que o dispositivo se encontra.

- **Sensor utilizado:** sensor de temperatura ambiente;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar, unidade;
- **Parâmetros de Saída:** temperatura em graus *Celsius*, temperatura em graus *Kelvin*, temperatura em graus *Farenheit*, temperatura em graus *Rankini*, temperatura em graus *Reaumur*, temperatura em graus *Romer*, temperatura em graus *Newton* e temperatura em graus *Deslile*;
- **Tipo:** Sensores.

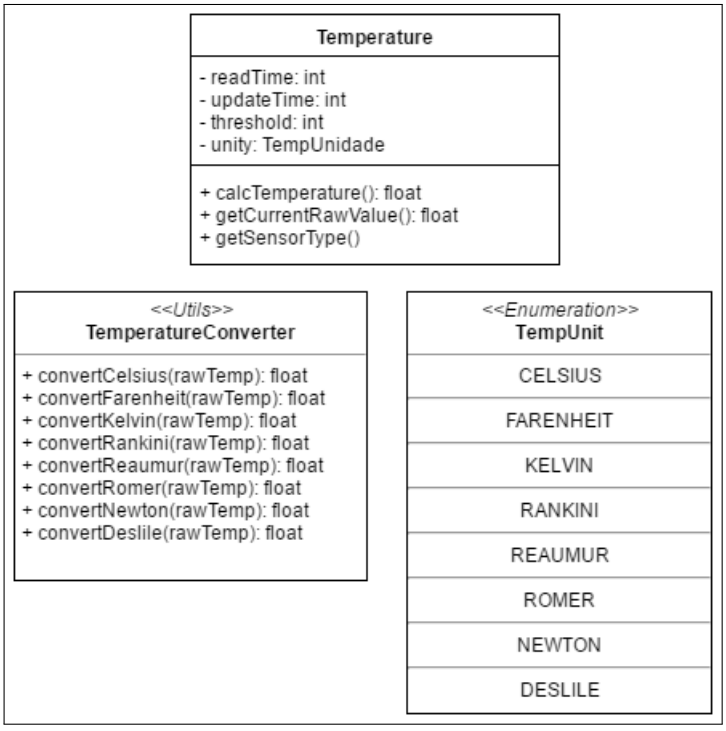


Figura 4.2: Diagrama de classes do Componente Temperatura.

Na Figura 4.2 ilustra-se o projeto orientado a objeto do Componente Temperatura. O parâmetro de entrada *tempo de leitura* define qual a duração que o sensor ficará ativo, ou seja, lendo as informações do meio. O *tempo de atualização* é utilizado para o componente atualizar a informação lida na sua última consulta ao sensor físico inerente ao dispositivo. O *limiar*, único parâmetro de entrada opcional, define um valor mínimo de interesse do usuário. Caso o sensor em questão leia do meio um valor abaixo do limiar esperado, este valor não será processado pelo componente e o retorno será nulo. A unidade define como será calculada a temperatura de retorno, podendo ser calculada em graus *Celsius, Farenheit, Kelvin, Rankini, Reamur, Romer, Newton* e *Deslile*, sendo estas representadas através do *Enumeration TempUnit*.

A classe Temperatura possui os seguintes métodos:

- *getSensorType()*: retorna o tipo do sensor, ou seja, a qual perfil este sensor pertence.
- *getCurrentRawValue()*: retorna o último valor lido pelo sensor de acordo com a última atualização, que funciona de acordo com o parâmetro de entrada *tempo de atualização*. Este método, por sua vez, é responsável por padronizar a leitura do sensor físico para possibilitar a interoperabilidade, ou seja, que o componente temperatura funcione independentemente do sistema operacional. Por exemplo, em um dispositivo de sistema operacional X seu sensor de temperatura captura os dados em graus *Celsius* e no sistema Y a captura, por padrão, é feita em *Farenheit*. Esta unidade de leitura precisa ser configurada ao instanciar este componente.
- *calcTemperature()*: internamente faz uso da classe estática utilitária *TemperatureConverter* para calcular a conversão do valor bruto capturado pelo sensor do dispositivo para a unidade especificada na entrada. Em síntese, é o método responsável por retornar a saída do componente, devidamente já convertida para a unidade de medida esperada. Sua implementação pode ser observada no Código 4.1.

Código Fonte 4.1: Implementação do método *calcTemperature()*.

```
1  switch(unidade){
2      case CELSIUS:
3          return TemperatureConverter.convertCelsius(
               getCurrentRawValue());
4      case FARENHEIT:
5          return TemperatureConverter.convertFarenheit(
               getCurrentRawValue());
6      case KELVIN:
7          return TemperatureConverter.convertKelvin(getCurrentRawValue
               ());
8      case RANKINI:
9          return TemperatureConverter.convertRankini(
               getCurrentRawValue());
10     case REAUMUR:
11         return TemperatureConverter.convertReaumur(
               getCurrentRawValue());
12     case ROMER:
13         return TemperatureConverter.convertRomer(getCurrentRawValue
               ());
14     case NEWTON:
15         return TemperatureConverter.convertNewton(getCurrentRawValue
               ());
16     case DESLILE:
17         return TemperatureConverter.convertDeslile(
               getCurrentRawValue());
18 }
```

Componente Proximidade *(Proximity)*

Componente que pode medir a proximidade de um objeto (por padrão, em centímetros) em relação à tela de um dispositivo. O objetivo deste componente é prover a distância existente entre o sensor de proximidade do dispositivo para um objeto qualquer em sua trajetória.

- **Sensor utilizado:** sensor de proximidade;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar, unidade;

- **Parâmetros de Saída:** distância do dispositivo para um objeto em centímetros ou distância do dispositivo para um objeto em milímetros ou uma variável de tipo booleana indicando se há algum objeto se deparando com o sensor de proximidade;

- **Tipo:** Sensores.

O sensor de proximidade é geralmente usado para determinar se um aparelho está sendo segurado próximo à orelha do usuário, permitindo determinar o quão longe um objeto encontra-se de um dispositivo. Muitos dispositivos retornam o dado bruto da distância absoluta (dada em cm), mas alguns retornam apenas valores que representam o nível de proximidade. Neste caso, o sensor normalmente relata seu valor máximo no estado longe e um valor menor no estado próximo (ex: 0 - próximo e 1 - longe).

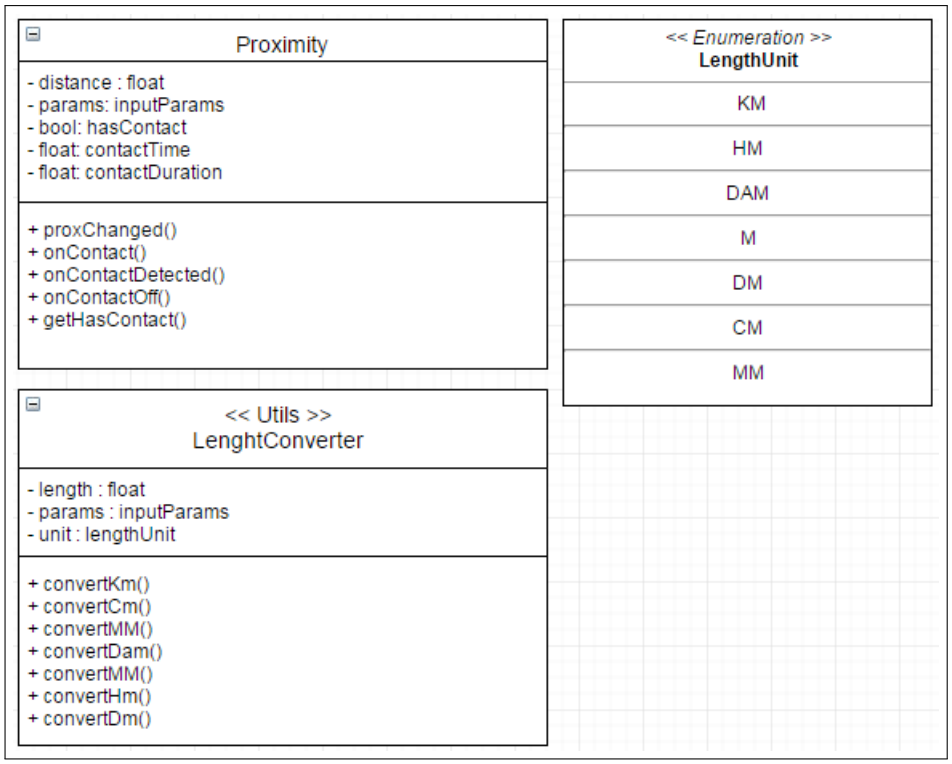


Figura 4.3: Diagrama de classes do Componente Proximidade.

No diagrama da Figura 4.3 podemos observar alguns métodos que estão disponíveis na classe *Proximity*. O método *onContact()* retorna se a distância capturada pelo sensor é a mínima possível, indicando que o dispositivo está em contato com algum outro corpo.

Componente Luz *(Luminosity)*

O componente luz encapsula a funcionalidade do sensor de luz, e tem como principal objetivo fornecer ao cliente o nível de iluminação apresentado no meio em que o dispositivo está inserido.

- **Sensor utilizado:** sensor de iluminação;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar, unidade;
- **Parâmetros de Saída:** valor da quantidade de luz na unidade lx (lux) chamado iluminamento; valor da quantidade de luz na unidade lm (lúmen), chamada fluxo; valor da quantidade de luz na unidade cd (candela), chamada intensidade; valor da quantidade de luz na unidade nit.
- **Tipo:** Sensores.

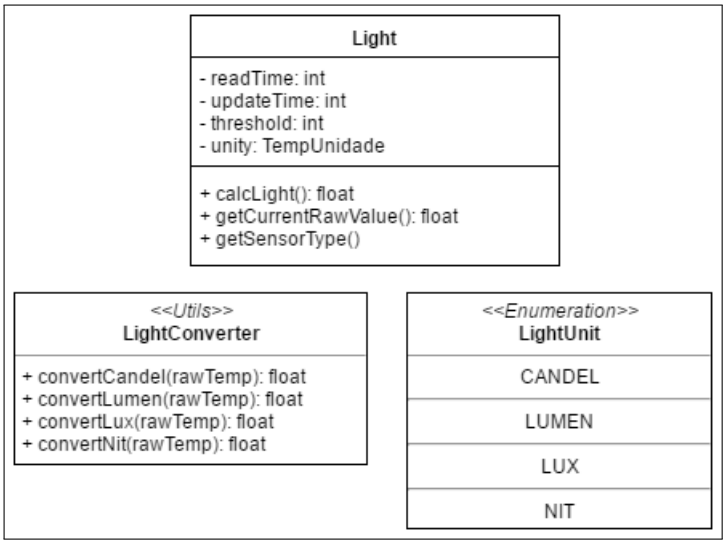


Figura 4.4: Diagrama de classes do Componente Luz.

Lux (lx) é a unidade SI (Sistema Internacional de Unidades de Medidas) de medida de iluminamento, que mede a incidência perpendicular de 1 lúmen em uma superfície de 1 metro quadrado.

Na Figura 4.4 podemos observar na classe *Luz* a definição dos métodos: *hasLuminosity()*, o qual verifica se há alguma iluminação no local levando em consideração o parâmetro de entrada limiar.

Componente Giroscópio *(Giroscopy)*

- **Sensor utilizado:** Sensor de Giroscópio;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar, unidade;
- **Parâmetros de Saída:** velocidade angular em cada eixo x, y e z;
- **Tipo:** Sensores.

Mede a velocidade angular em três dimensões em diferentes unidades de medida angular (ex: rad/s, graus/s) em volta dos eixos x, y e z do dispositivo.

O sensor giroscópio faz a detecção de movimentos, em particular de giros e inclinações. Na Figura 4.5 é possível ver o diagrama de classes do sensor Giroscópio. Normalmente, a variação angular é extraída da velocidade, através da sua multiplicação pelo intervalo de tempo (At) entre a atual e a última saída, gerando um incremento de rotação que, integrado ao longo do tempo, permite a formação do vetor de orientação. Comparado-o ao acelerômetro, suas respostas são muito rápidas às mudanças de ângulos, embora necessite de maior potência para operar.

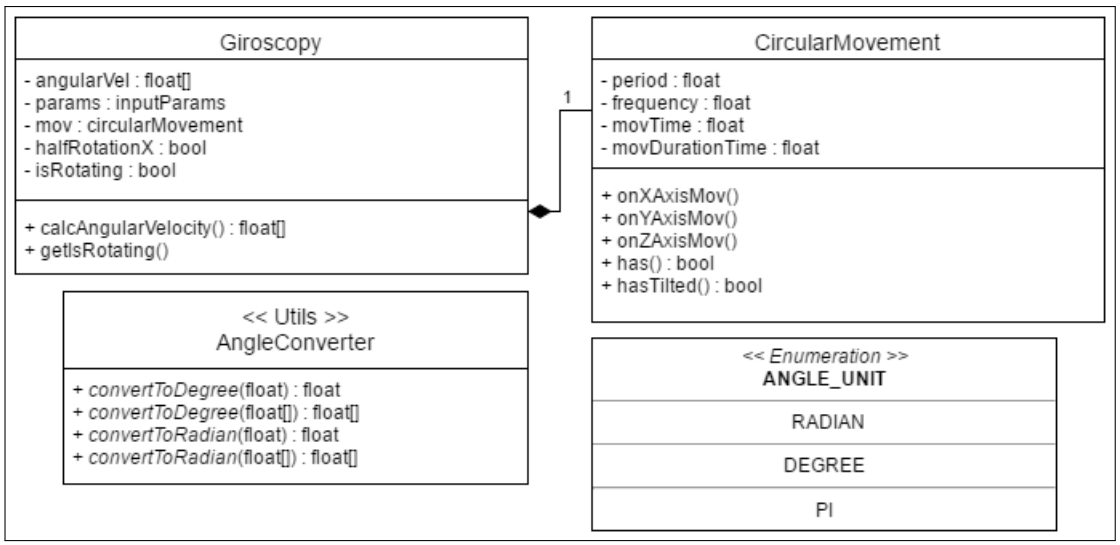


Figura 4.5: Diagrama de classes do Componente Giroscópio.

Componente Gravidade *(Gravity)*

- **Sensor utilizado:** sensor de gravidade;

- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar da força nos três eixos (x, y, z);
- **Parâmetros de Saída:** Valor da força de aceleração aplicada pela gravidade em cada eixo x, y e z;
- **Tipo:** Sensores.

Este é um dos componentes de sensores mais simples e mede a força da gravidade aplicada ao dispositivo em todos os três eixos físicos (x, y, z). De modo geral, pode ser utilizado, para além da detecção da gravidade, na detecção de movimentos específicos do dispositivo tais como vibrações ou inclinações. Neste contexto, a depender da *API* nativa utilizada ou dispositivo, a força de gravidade mais intensa em determinado eixo obtida pode ser representada por valores máximos (e vice-versa), representando deste modo tais movimentações.

Na Figura 4.6 podemos observar na classe *Gravity* a definição dos métodos: *hasTiled()*, o qual verifica se houve uma inclinação no dispositivo, *hasShaked()*, que em sua implementação utiliza o método anterior investigando se houveram várias inclinações em um curto espaço de tempo. A classe possui também o método *getIsOnGround()*, o qual tem por objetivo reconhecer se o dispositivo está em repouso.

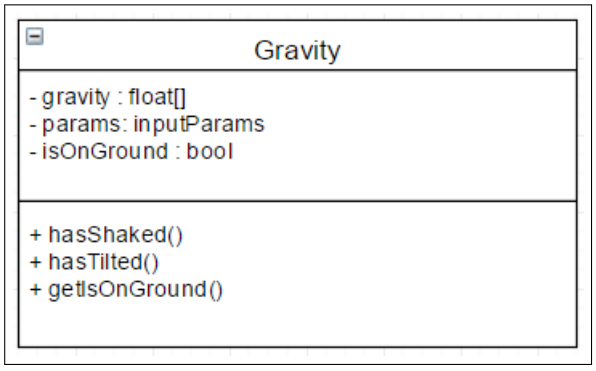


Figura 4.6: Diagrama de classes do Componente Gravidade.

Componente Orientação *(Orientation)*

- **Sensor utilizado:** Sensor de Orientação;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar do ângulo nos três eixos x, y, z;

- **Parâmetros de Saída:** Valor do ângulo aplicada em cada eixo x, y e z;
- **Tipo:** Sensores.

Mede os graus de rotação (ângulo) que um dispositivo realiza em torno de todos os três eixos físicos. A orientação de um dispositivo é baseada em ângulos de rotação tomando-se como referência algum quadro de referência, estruturado sobre um sistema de coordenadas fixo. Calculando a orientação de um dispositivo, pode-se monitorar a posição do dispositivo em relação ao quadro da referência da Terra (especificamente, o polo norte magnético). O sistema calcula os ângulos de orientação usando sensor de campo magnético de um dispositivo em combinação com o acelerômetro do dispositivo.

No diagrama de classes da Figura 4.7, podemos ver como se dá a organização das classes dentro do componente orientação, a classe de utilitários *AngleVectorConversor* apresenta os seguintes métodos auxiliares: *convertToQuaternion()*; *convertToEulerAngle()*; *convertToMatrix()* todos eles utilizados para converter a representação do ângulo internamente.

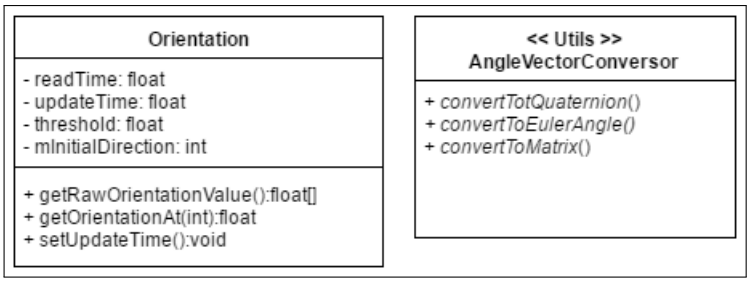


Figura 4.7: Diagrama de classes do Componente Orientação.

Componente Acelerômetro *(Accelerometer)*

Este componente encapsula a funcionalidade do sensor de acelerômetro, que mede a aceleração aplicada nas três dimensões x, y e z do dispositivo. Através deste sensor, é possível avaliar a posição relativa do aparelho e utilizar esse dado para ajustar o visor do celular ou *tablet*.

- **Sensor utilizado:** Sensor de Acelerômetro;
- **Parâmetros de Entrada:** tempo de leitura, tempo de atualização, limiar definido para cada eixo x, y e z (parâmetro opcional);

- **Parâmetros de Saída:** Valor da força de aceleração aplicada em cada eixo x, y e z;
- **Tipo:** Sensores.

Se o dispositivo estiver estendido em uma superfície horizontal plana, a aceleração no eixo Z deverá ser de aproximadamente 9,8 m/s^2, devido à força da gravidade atuando sobre o aparelho.

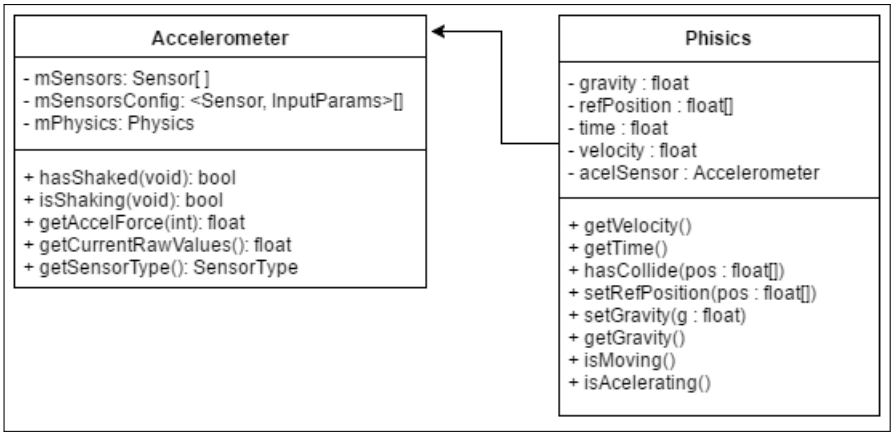


Figura 4.8: Diagrama de classes do Componente Acelerômetro.

Os métodos *hasShaked()* e *isShaking()* mostrados na Figura 4.8 são usados para verificar se o dispositivo sofreu uma agitação ou se está sendo agitado naquele exato momento respectivamente, e retornam uma variável booleana para o desenvolvedor programar algum ação para este comportamento reconhecido.

4.1.2 Componentes de Gerenciamento *(PervasiveSensorManager)*

Este componente é de suma importância, pois ele se comunica diretamente com a *API* nativa do sistema operacional, ou seja, uma instância deste componente irá implementar as funções nativas do sistema ao qual ele pertence. Neste sentido, todos os componentes de sensores são instanciados através do Componente de Gerenciamento o qual já possui conhecimento acerca do sistema operacional corrente, e trata os dados particulares de acordo com com a plataforma na qual sua instância foi criada. Mantém-se assim o requisito básico de interoperabilidade nos componentes [32].

- **Parâmetros de Entrada:** Sistema Operacional, lista de Componentes do tipo Sensor para cadastro;

- **Parâmetros de Saída:** Atualização dos dados de cada componente de sensor cadastrado;

- **Tipo:** Componente de Gerenciamento.

O componente *PervasiveSensorManager* tem como responsabilidade implementar as funções de gerenciamento dos componentes de sensores, ou seja, funções que cadastram os sensores que a aplicação tem interesse em coletar, funções que alteram o tempo de atualização de cada sensor, funções que verificam o suporte daquele dispositivo para com o sensor em questão, funções que descadastram sensores, entre outras. Além disso, tal gerenciador mantém o grupo de parâmetros que são considerados relevantes e comuns a todos os sensores pois definem parte do modelo de componente, no que diz respeito ao sensoriamento. Todos os componentes de sensores devem implementar o *InputParams* para manter o contrato que foi definido para os componentes poderem se encaixar. São exemplos de tais parâmetros: tempo de leitura, tempo de atualização, limiar e unidade, como podem ser vistos na Figura 4.9.

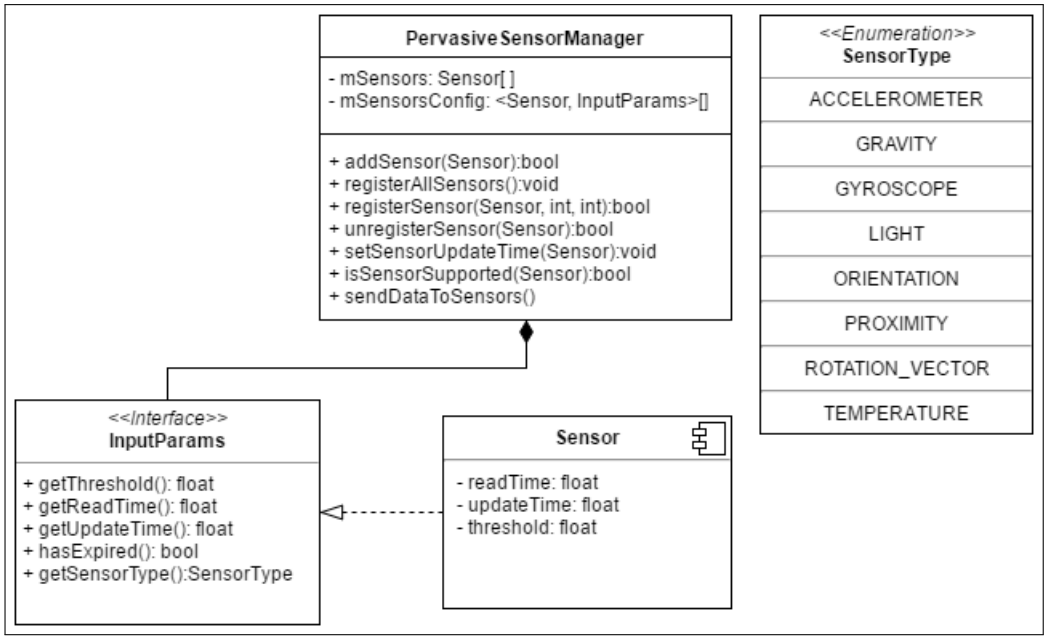


Figura 4.9: Diagrama de classes do componente Gerenciador.

4.1.3 Componentes do tipo Interação *(Interaction)*

Os componentes de interação são aqueles responsáveis pela conexão com o sistema cliente (ex: componentes, aplicação cliente, arcabouços de *software*). Neste sentido, foram considerados dois tipos principais, a depender do escopo de atuação das interações entre tais entidades, tais como: interações internas (Componentes de Comunicação) e interações externas ou comunicação da biblioteca com os clientes (Componente de Ponte).

1. **Componente de Ponte *(Bridge)***

   Este componente tem como entradas o sistema operacional, um identificador da aplicação, componentes desejados (se Componente de Domínio ou de Sensor) e seus respectivos parâmetros, e caso necessário, as operações a serem realizadas pelo Componente de Domínio.

   Este componente é responsável pela comunicação entre os componentes pertencentes à biblioteca com componentes externos, presentes na aplicação cliente. Este componente fornece uma interface, que deve ser o único meio de acesso externo, provendo desta forma autenticação dos clientes e delegação de permissões específicas. Além disto, requisitos da aplicação externa relacionados a aspectos de plataforma (ex: sistema operacional, estruturas de dados específicos, unidades de medida) são passadas como parâmetro. Deste modo, é facilitado para os componentes internos à biblioteca, a definição de dados e execução de rotinas particulares de cada plataforma operacional. Além disto, tais informações tornam-se acessíveis a todos os componentes internos.

2. **Componente de Comunicação *(Comunication)***

   Responsável por fazer ponte (genérica) entre componentes. Recebe como parâmetro um vetor de componentes sensores que irá compor uma "rede de sensores". Atua como um "servidor", que recebe dados de cada um dos sensores pertencentes ao vetor, realiza processamentos específicos e os compartilha, quando necessário. Ex: Este componente pode ser utilizado em casos em que a precisão de um único sensor não é adequada, como por exemplo para o cálculo de velocidade. Nestes casos, pode ser usado no Conjunto de Componentes *KINEMATICS*, a fim de realizar a comunicação entre os sensores de tal Domínio (Acelerômetro e GPS).

Pode-se ter casos em que para validar um valor (ex: gravidade) recebe-se tais valores de mais de uma fonte (ex: sensor de gravidade, acelerômetro, e aceleração linear), e calcula-se uma média entre todos eles, retornando esta média para todos os sensores.

4.1.4 Componente de Domínio *(Domain)*

Os Componentes de Domínio são os únicos componentes da biblioteca que agrupam outros componentes, além de pré-definir um conjunto de componentes de sensores. Uma vez descritos todos os componentes propostos pelo modelo, ilustra-se como o cliente (desenvolvedor) pode utilizá-los em conjunto, a fim de representar determinado contexto dos usuários de suas aplicações pervasivas.

- **Entrada:** domínio e operações desejadas (enum).
- **Saída:** retorno das operações desejadas.
- **Tipo:** Componente de Domínio.

Neste sentido, para exemplificar, foram concebidos quatro possíveis cenários de representação do contexto do usuário, sendo estas apresentadas na Figura 4.10. O desenvolvedor pode utilizar: (A) apenas 1 componente de sensor; (B) um grupo de sensores (independente de domínios específicos); (C) um conjunto de sensores agrupados num Componente de Domínio; ou (D) um conjunto de Componentes de Domínio.

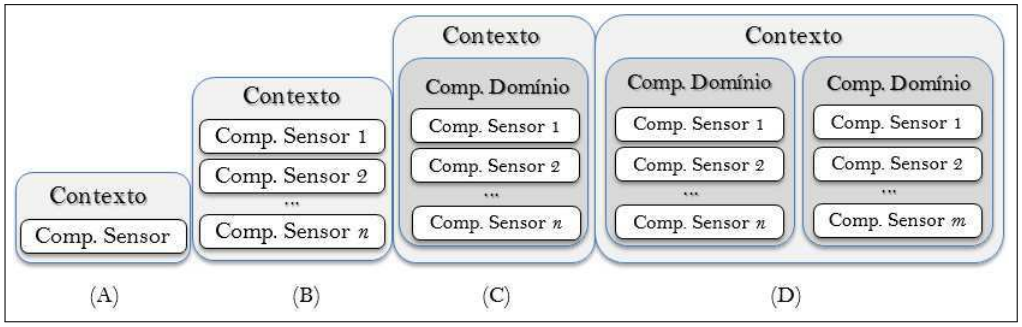


Figura 4.10: Exemplo de cenários de representação de contexto.

No caso (A), pode-se representar um contexto com apenas um único sensor, exemplificando: O sensor de luz pode ser empregado para reconhecer se está de dia ou de noite, caso a luminosidade do meio, capturada através do sensor de luminosidade, fosse baixa assume-se

que está de noite. Porém, esta seria uma dedução ineficiente, pois se o sensor estiver inserido em um ambiente fechado o qual possui uma iluminação artificial, sendo noite, o sistema iria deduzir que era dia equivocadamente. Utilizando o caso (C) ou (D), que fornece uma interface para operações com sensores dentro de um mesmo domínio, de modo a auxiliar ao desenvolvedor na concepção de cenários de aplicações diversas (ex: aplicações de multimídia, geoprocessamento e movimentação), a representação de um contexto se torna mais eficiente. Domínios são grupos de sensores que utilizam seus dados em prol de cálculos mais apurados em relação ao tipo de contexto ou ambiente a ser reconhecido em questão. Alguns exemplos dos domínios pré-definidos são ilustrados na Tabela 4.1.

De modo objetivo, uma aplicação que se utiliza do domínio pré-definido *MULTIMIDIA*, onde são agrupados os sensores de áudio (microfone) e vídeo (câmera), pode ser utilizado em conjunto com instâncias de outros Componentes de Domínio para o fornecimento de detalhes de contexto específicos. Neste sentido, componentes dos domínios *KINEMATICS* e/ou *LOCATION* podem ser utilizados em conjunto com o componente de *MULTIMIDIA*, a fim de prover o reconhecimento de padrões de voz e vídeo, em movimento, por exemplo.

Tabela 4.1: Exemplos de Domínios predefinidos.

Contexto	**Descrição**	**Sensores**
IN-OUT-DOOR	Grupo de sensores que permitem identificar se o usuário está em ambiente fechado ou aberto.	Termômetro, Luminosidade, Campo Magnético
KINEMATICS-1	Aplicações que envolvam aspectos de movimentação (uniforme ou variada) em um pequeno raio.	Acelerômetro, Orientação, Aceleração Linear, Gravidade
KINEMATICS-2	Aplicações que envolvam aspectos de movimentação (uniforme ou variada) em raios maiores (ex: esportes).	Acelerômetro, Orientação, Aceleração Linear, Gravidade, GPS
LOCATION	Sensores de localização.	GPS, Orientação
COLLISION	Detecção de queda do dispositivo.	Pressão, Gravidade, Acelerômetro, Aceleração Linear
CLIMATE	Sensores que permitem sensibilidade ao clima de determinadas regiões.	Termômetro, GPS, Umidade relativa, Pressão
INTERACTION	Reúne sensores para prover interação natural homem-máquina. Pode ser utilizado em aplicações onde o dispositivo sirva como controle (ex: jogos 3D, ensino).	Orientação, Rotação, Proximidade, Pressão, Giroscópio, Luminosidade
CLOUD	Reúne os sensores que permitem conectividade.	Bluetooth, wi-fi, 3G
MULTIMÍDIA	Reúne os sensores para aplicações multimídia, como reconhecimento de voz e/ou face, entre outros.	Camera, microfone

A relação entre os componentes supracitados é exibida através da Figura 4.11, a qual mostra cada componente posicionado de acordo com sua função dentro da abordagem. O componente de Ponte é o responsável por receber os requisitos diretamente da aplicação e direcioná-los para cada componente específico do sistema. Podendo empregar o componente de Domínio ou não, o componente Gerenciador irá repassar os parâmetros de entrada dos sensores para os próprios componentes de Sensores por meio do componente de Comunicação, que é responsável por fazer essa relação.

Por fim, quando os componentes de Sensores realizarem seus trabalhos, o processamento será feito nos componentes de Comunicação e/ou Domínio para serem retornados a aplicação final através dos componentes de Gerenciamento e Ponte.

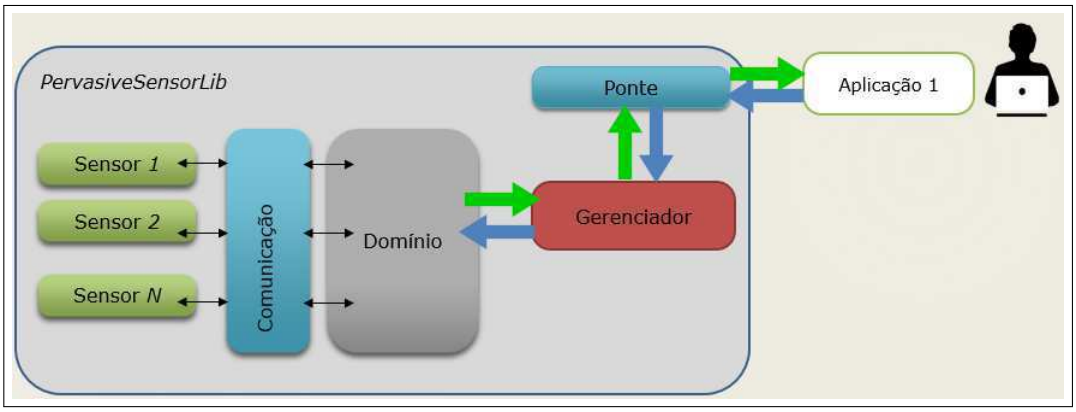


Figura 4.11: Representação da relação entre os componentes.

4.2 *SensMonitors*: Aplicação Protótipo de Validação do Modelo

Nesta seção, é apresentado um exemplo de uso da *PervasiveSensorLib*, buscando um melhor entendimento de como o modelo proposto pode trazer vantagens no desenvolvimento das aplicações pervasivas. Uma vez definido o modelo de componentes da biblioteca, e após o seu desenvolvimento, foi definido uma aplicação de teste, a fim de validar o modelo e servir como base para a realização dos experimentos de avaliação com grupos de desenvolvedores, que se encontra detalhada no Capítulo 5. Partindo do pressuposto de que o contexto de aplicações pervasivas se baseia em grande parte nos dados captados pelo uso de sensores,

buscando um cenário do mundo real, onde as aplicações devem capturar as características do meio, para a partir deste ponto, tomar decisões e/ou sugerir ações com base nos dados obtidos.

A aplicação protótipo permite que o usuário navegue em uma lista de abas para a consulta dos valores coletados em tempo-real por cada sensor, conforme exibido nas Figuras 4.12 e 4.13. Optou-se pela utilização de uma *interface* de interação simples e intuitiva. As tecnologias utilizadas para o desenvolvimento do código foram: *IDE Android Studio v. 1.4* e dispositivos com *Android 4.4 (Kitkat)* e *Android 5.0 (Lollipop)*.

A aplicação foi projetada com base nos seguintes requisitos principais: instanciar os componentes de sensores (4.1.1), cadastrá-los e gerenciá-los devidamente através do componente de gerenciamento (*PervasiveSensorManager*) (4.1.2), passar e obter os parâmetros necessários ao modelo, ser sensível às adaptações e atualizações dos valores obtidos, dentre outros.

Para o desenvolvimento da aplicação, o exemplo de código no site[1] da *API Android* foi utilizado como base. Após importar a biblioteca *PervasiveSensorLib* algumas alterações pontuais foram elaboradas no código exemplo, além da implementação de *fragments*. Um *Fragment* representa o comportamento ou uma parte da *interface* do usuário em uma atividade implementada. Torna-se possível combinar vários fragmentos de *GUI* em uma única atividade para compilar uma lista de painéis múltiplos e reutilizar um fragmento em diversas atividades.

Neste sentido, um *fragment* é como uma seção modular de uma atividade, que tem o próprio ciclo de vida, recebe os próprios eventos de entrada e que pode ser adicionado ou removido com a atividade em execução (uma espécie de "sub-atividade"que pode ser reutilizada em diferentes atividades). Neste sentido, a classe *SensorFragment*, possui uma instância do componente de gerenciamento *PervasiveSensorManager*.

Como o requisito da aplicação foi testar o uso dos sensores e atualização dos valores obtidos, com auxílio do componente de gerenciamento, optou-se por não utilizar um Domínio em específico, mas sim, cadastrar diretamente cada sensor. Uma vez detectada a mudança de valores obtidos, é processada a função de *callback* para atualização dos dados de modo adequado. Na Figura 4.12, é apresentada a primeira delas na qual deveria ser demonstrado o

[1] https://developer.android.com/samples/SlidingTabsBasic/project.html

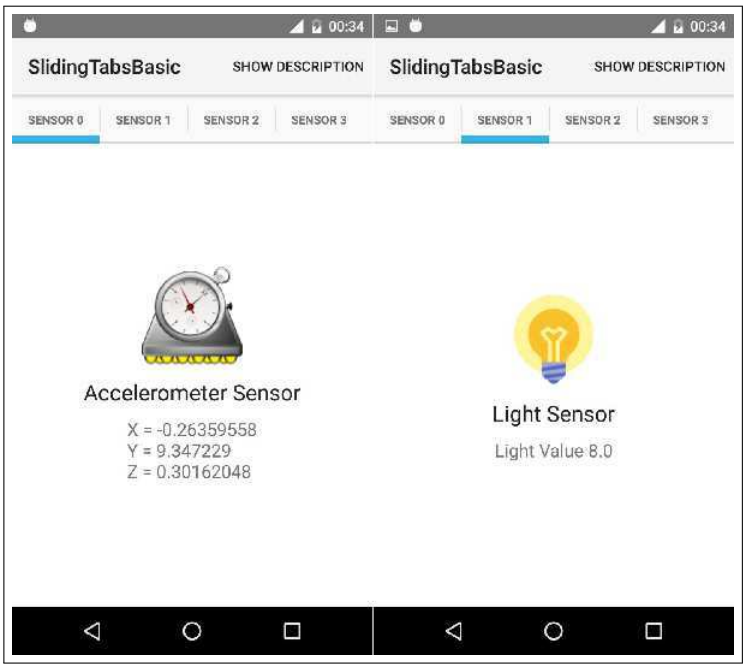


Figura 4.12: Abas 0 e 1, Sensor Acelerômetro e Sensor de Luz.

uso do sensor acelerômetro. Ainda na Figura 4.12, é ilustrada também a segunda aba, onde os dados a serem mostrados são do componente luz. Nesta aba, foi implementada a condição para quando o parâmetro de saída retornasse um valor inferior ao limiar definido de 2 unidades de iluminação, o ícone da lâmpada acesa, deveria ser substituído por outra imagem representando uma lâmpada apagada.

Na Figura 4.13, tem-se a aba do componente de orientação, onde a inclinação da imagem na tela do dispositivo, deveria ser orientada de acordo com o ângulo capturado pelo sensor do dispositivo. O ícone funcionava como uma bússola que apontava sempre para o valor recebido através do parâmetro de saída do componente.

Ainda na Figura 4.13, tem-se a última aba do protótipo, onde deveria ser demonstrado o uso do componente de proximidade. Nesta tela, a lógica implementada era a de alterar o ícone de proximidade de acordo com o que estava sendo capturado pelo sensor no momento. O valor do parâmetro de saída, distância, quando igual a 0 (zero) indica estar próximo. Caso contrário, o ícone mostrado representava longa distância.

Com base no desempenho da aplicação gerada, foi observado que o componente gerenciador de fato auxilia na instanciação dos componentes de sensores, encapsulando para o desenvolvedor, os aspectos que dizem respeito ao tratamento de dados em baixo nível. A

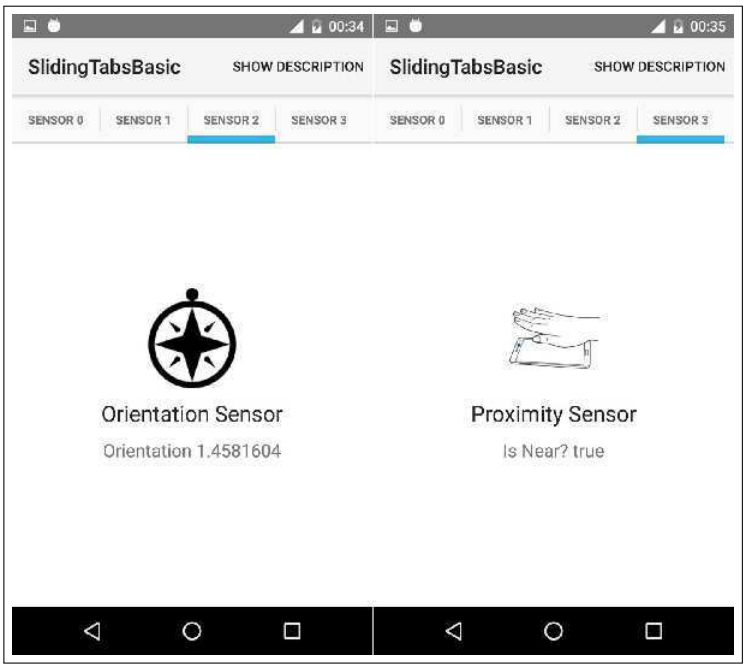


Figura 4.13: Abas 2 e 3, Sensor de Orientação e Sensor de Proximidade.

fim, de verificar o desempenho de outros desenvolvedores com experiência em *Android* ao utilizar a biblioteca, foi realizada um experimento de avaliação, com base nesta aplicação.

Capítulo 5

Análise dos Resultados

Neste capítulo, é descrita a metodologia utilizada durante os experimentos, bem como uma avaliação dos resultados obtidos. Através dos experimentos, buscou-se verificar o quanto a abordagem proposta colabora para o aumento na produtividade do desenvolvimento de aplicações pervasivas e da leitura do contexto dos usuários. A metodologia do estudo de caso se deu através dos seguintes passos: delineamento do estudo (plano de execução, método de análise), definição do perfil dos participantes e divisão dos grupos, definição das métricas de avaliação, definição do protótipo a ser desenvolvido e, por fim, a coleta dos dados para serem analisados.

Como o experimento limitou-se ao desenvolvimento de um protótipo, ele foi especificado de forma que fosse necessário o uso de sensores, buscando um cenário do mundo real, onde as aplicações devem capturar as características do meio, para a partir deste ponto, tomar decisões e/ou sugerir ações com base nos dados obtidos. Um projeto base, em conjunto com um tutorial sobre a biblioteca foram disponibilizados para os programadores trabalharem, além de uma especificação da aplicação que deveria ser desenvolvida. O projeto base possuía algumas classes que eram necessárias para a implementação das abas presentes no protótipo e sua função era direcionar todo o trabalho dos programadores, permitindo focar apenas no uso de sensores. Os desenvolvedores precisaram programar em uma única classe para fazerem o que lhes era pedido. Esta classe era chamada *SensorFragment*, a qual era um *Fragment* pertencente a *API* da plataforma *Android*.

Para a realização do experimento, foram recrutados dez alunos voluntários do curso de Ciência da Computação que cursavam acima do 3º período e que eram participantes do pro-

jeto de capacitação da Sony no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded), o qual fica localizado na Universidade Federal de Campina Grande (UFCG). O projeto de capacitação da Sony trabalha com o desenvolvimento de soluções *Android*. Deste modo, garantiu-se que todos os integrantes tivessem conhecimento no desenvolvimento para a plataforma *Android*, tornando possível suas participações no estudo de caso, onde seria necessário implementar o protótipo em *Android*.

Os dez desenvolvedores voluntários foram divididos em dois grupos de cinco pessoas. Para cada grupo houve um encontro, ambos estimados em duas horas de duração. O horário escolhido para a elaboração do estudo de caso foi entre 18 horas e 20 horas, pois os voluntários estavam disponíveis, além de diminuir o risco de possíveis interrupções que são comuns durante o horário de trabalho. O local da execução das experiências foi o próprio laboratório de capacitação, pois simplificaria o processo de atribuir máquinas e dispositivos para cada integrante. O motivo da separação em dois grupos se deu com o intuito de eliminar o viés do aprendizado, pois, cada grupo foi responsável por desenvolver a mesma aplicação, a diferença foi que um dos grupos utilizou a *PervasiveSensorLib* em seu desenvolvimento e o outro grupo utilizou a própria *API* nativa da plataforma *Android*.

Nenhum dos desenvolvedores conhecia a biblioteca em estudo, *PervasiveSensorLib*, e só tiveram acesso à documentação durante a execução do experimento. Todos os códigos escritos pelos desenvolvedores voluntários podem ser encontrados no Apêndice C. Nesta ocasião, os programadores tiveram acesso apenas à documentação *Android* para uso dos sensores em suas implementações.

Cada integrante tinha seu posto dentro da sala do laboratório onde as máquinas possuíam sistema operacional Ubuntu 14.04, com 2GB de memória RAM. Os programadores utilizaram a IDE *Android Studio* Versão 1.4, e dispuseram dos dispositivos, com versões *Android* 4.4 (Kitkat) e *Android* 5.0 (Lollipop) ambas oferecidos pelo projeto de capacitação. O segundo encontro, ocorreu com os outros 5 integrantes. Desta vez, os programadores tiveram o auxílio da biblioteca em estudo para a implementação do protótipo.

Em relação ao projeto da pesquisa, foram utilizadas diferentes metodologias para atender ao objetivo da mesma, tais como um estudo de caso comparativo e um *survey*. Segundo Wohlin, C e Runeson, P. [36], o estudo de caso é um dos métodos mais utilizados em ciências como a Medicina e a Computação, e tem como foco o estudo aprofundado e intenso

de uma unidade no seu ambiente natural, a partir de diversas fontes de evidência, pelo emprego de diversas técnicas de coleta de dados. Para tanto, é fundamental que: a) não haja manipulação dos indivíduos pertencentes às fontes de evidência; e que b) o pesquisador não exerça controle nenhum sobre os indivíduos. Neste sentido, no Estudo de Caso Comparativo são expostos diferentes grupos a tratamentos distintos buscando-se determinar e quantificar o relacionamento entre as variáveis envolvidas.

O processo de avaliação foi baseado nos conceitos de uma abordagem bastante adotada na engenharia de *software* para colher opiniões de desenvolvedores após o uso de uma linguagem e/ou ferramenta, chamada *survey* [36]. O *Survey* baseia-se na aquisição de dados da memória de pessoas sujeitas a um questionário ou entrevista.

Em relação aos resultados obtidos no *Survey*, foi visto na Figura 5.1, a qual mostra as respostas dos desenvolvedores do primeiro experimento, que calculando a média ponderada, o nível de dificuldade obtido pelos voluntários se mostrou igual a 3,0, onde o máximo de dificuldade possível era 5,0. No outro experimento, presente na Figura 5.2, o valor da média ponderada obtido se mostrou igual a 1,8. Logo, de acordo com os gráficos mostrados, podemos concluir que, a turma 1, em geral, avaliou a dificuldade de implementação em 60% e a turma 2 avaliou a dificuldade em 36%. Pode-se concluir que houve um ganho na facilidade ao desenvolver utilizando a biblioteca sugerida em comparação com o desenvolvimento utilizando apenas a API nativa do android.

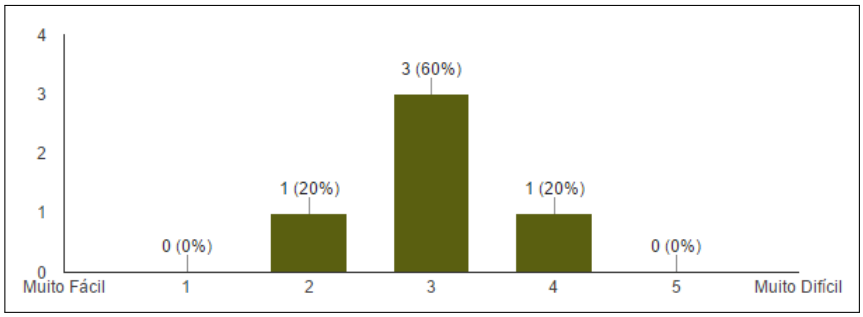


Figura 5.1: Grau de dificuldade do experimento turma 1.

Métricas de *software*, do ponto de vista de medição, podem ser divididas em duas categorias: medidas diretas e indiretas. Podemos considerar como medidas diretas do processo de engenharia de *software* o custo e o esforço aplicados ao desenvolvimento e manutenção do *software* e do produto, a quantidade de linhas de código produzidas e o total de defeitos

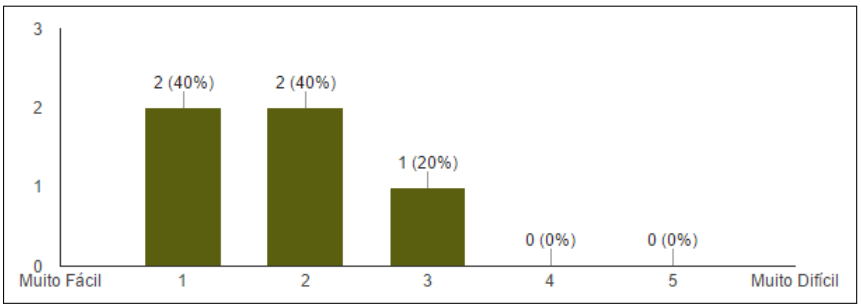


Figura 5.2: Grau de dificuldade do experimento turma 2.

Tabela 5.1: Métricas de cada desenvolvedor no segundo experimento.

Voluntário	**Tempo Total**	**Número de Linhas**
Desenvolvedor 1	58 min	132
Desenvolvedor 2	52 min	158
Desenvolvedor 3	40 min	127
Desenvolvedor 4	44 min	117
Desenvolvedor 5	55 min	126

registrados durante um determinado período de tempo. Porém, a qualidade e a funcionalidade do *software*, ou a sua capacidade de manutenção, são mais difíceis de serem avaliadas e só podem ser medidas de forma indireta.

As métricas definidas para o estudo de caso foram: tempo de codificação e o número total de linhas de código. A coleta dos dados referentes às métricas escolhidas dos experimentos ocorreu ao término da implementação. A captura do tempo gasto por desenvolvedor foi realizada salvando o tempo de início no momento em que as máquinas dos integrantes estavam aptas para iniciar o desenvolvimento, isto é, contendo o projeto importado na IDE *Android Studio* e o acesso à API necessária, além de um dispositivo conectado a cada computador, facilitando o processo de depuração. O protótipo implementado podia ser julgado como finalizado caso fosse aprovado em um conjunto de testes visuais e manuais realizados a pedido de cada participante.

O experimento resultou em cinco protótipos desenvolvidos com a API nativa do *Android* e outros cinco protótipos desenvolvidos com a biblioteca *PervasiveSensorLib*, todos disponíveis nas seções Apêndice C e Apêndice D. As Tabelas 5.2 e 5.1 mostram o tempo e o

Tabela 5.2: Métricas de cada desenvolvedor no primeiro experimento.

Voluntário	Tempo Total	Número de Linhas
Desenvolvedor 1	1 h 06 min	155
Desenvolvedor 2	1 h 31 min	143
Desenvolvedor 3	1 h 40 min	184
Desenvolvedor 4	1 h 52 min	158
Desenvolvedor 5	1 h 44 min	161

Tabela 5.3: Média total dos resultados obtidos

Experimento	Duração Média	Média de Linhas de Código
1	1 h 35 min	160,2 linhas
2	50 min	132 linhas

número de linhas de código de cada desenvolvedor, separando os que utilizaram API nativa dos que utilizaram a biblioteca, respectivamente. Da Tabela 5.3 podemos concluir que houve uma diminuição média de aproximadamente 29 linhas de código, se tratando apenas de uma classe editada por cada participante, devido ao código base ter sido oferecido, este valor se torna significativo. Quanto ao tempo médio, também obtivemos uma redução importante, que para o desenvolvimento de aplicações semelhantes, a longo prazo possivelmente traria uma redução de custo na produção desse tipo de *software*.

Capítulo 6

Conclusões

Neste capítulo, são apresentadas as conclusões e as limitações do trabalho desenvolvido, bem como as sugestões de pesquisas a serem exploradas futuramente. Como observado ao longo do trabalho, modernos paradigmas computacionais relacionados à mobilidade e ciência de contexto apresentam novas demandas e requisitos a serem atendidos adequadamente [6], [29]. Por serem áreas de pesquisa relativamente novas, observa-se que muitos pontos permanecem em aberto à investigação e solução pelas comunidades científicas e industriais.

Foi observado durante a pesquisa que existem características específicas de sistemas pervasivos e que normalmente não são consideradas pelas abordagens de desenvolvimento tradicionais da engenharia de *software*, tais como: abstração das complexidades envolvidas, heterogeneidade de dispositivos, adaptabilidade, dentre outras. Desta forma, ao utilizar tais metodologias para o desenvolvimento de *software* pervasivo, é possível que sua eficiência e/ou eficácia não atinjam o desempenho esperado [14].

Diversas aplicações necessitam processar dados de sensores continuadamente a fim de determinarem contexto/atividade dos seus usuários, e eventualmente ativar outros eventos [31]. Neste sentido, apesar de estas aplicações poderem ser implementadas por sistemas de inferência diversos, muitas vezes torna-se demasiadamente custoso o desenvolvimento de rotinas necessárias durante e após sensoriamento (ex: detecção de movimentos, posições). Alguns problemas principais são apontados por Ravindranath et. al. [25], dentre os quais destacamos dois: 1) Abstrações de *software* pobres e 2) carência de *APIs* como suporte ao desenvolvimento. No primeiro caso, segundo o autor supracitado, diversas *APIs* existentes são dependentes de tecnologias de *hardware*, tornando necessário o conhecimento de pro-

gramação de baixo nível para o tratamento dos dados de sinais captados pelos sensores. No segundo caso, funcionalidades potencialmente reutilizáveis por diferentes tipos de aplicações muitas vezes não são providas pelas *APIs* de sensoriamento de dados existentes e acabam implicando em maior custo de desenvolvimento. Como exemplo, rotinas para distribuição de dados dos sensores, aspectos como segurança de acesso, garantia de conectividade, ainda são requisitos que impactam na produtividade de desenvolvimento [27].

Observadas estas problemáticas, este trabalho apresentou uma infraestrutura baseada em componentes para facilitar o desenvolvimento de aplicações pervasivas. A infraestrutura proposta disponibiliza um conjunto de componentes que permite a reutilização das funcionalidades implementadas em diversas aplicações, visando a otimização de desenvolvimento. Tal solução baseia-se no paradigma de Desenvolvimento Baseado em Componentes (DBC), sendo especificados diferentes conjuntos de componentes de acordo com um modelo de componente definido.

Considera-se como principal contribuição do trabalho o fruto de tais análises e a aplicação dos conceitos relacionados à metodologia de DBC (encapsulamento e componentização) a fim de otimizar o desenvolvimento de aplicações pervasivas. Como resultado principal, apontamos a proposta de uma biblioteca de componentes de sensores (*PervasiveSensorLib*) objetivando a redução no esforço de desenvolvimento através do reúso e possibilitando o aumento na produtividade. Além da abordagem proposta, representada pelo Modelo de Componentes definido, a implementação da biblioteca para dispositivos *Android* validou a abordagem e demonstrou a sua viabilidade técnica. A biblioteca foi utilizada no desenvolvimento de aplicações pervasivas, o que respalda ainda mais a utilidade do resultado obtido.

Para validação, foi realizado um experimento de avaliação das vantagens práticas obtidas, onde foi comparada a produção de uma aplicação protótipo (*PervasiveSensorLib* com a *API* nativa do *Android*). Para avaliar a produtividade dos grupos, diferentes métricas foram usadas: o número de linhas de código e o tempo de desenvolvimento (horas).

Tal experimento permitiu-nos perceber limitações e possibilidades de expansões futuras. Com relação às limitações do trabalho, apontamos a dificuldade encontrada na realização de experimentos com um grupo maior de desenvolvedores, primeiramente devido ao próprio escopo e tempo do projeto e pela carência de alunos (público-alvo) capacitados adequadamente para o desenvolvimento em *Android*. Tal limitação do estudo apresenta uma possibilidade

de trabalho futuro, podendo ser realizadas avaliações comparativas com grupos maiores de indivíduos, onde possam ser exploradas métricas mais subjetivas (medidas de *software* indiretas), também relacionadas à qualidade final dos produtos gerados com a biblioteca *PervasiveSensorLib*. A Utilização de pessoas com um certo grau de conhecimento no que se trata de sensores para o experimento também seria uma abordagem interessante para verificar se ainda haveria ganho utilizando a abordagem proposta.

Bibliografia

[1] Javier Andreu-Perez, Daniel R Leff, HMD Ip, and Guang-Zhong Yang. From wearable sensors to smart implants—toward pervasive and personalized healthcare. *IEEE Transactions on Biomedical Engineering*, 62(12):2750–2762, 2015.

[2] Feruzan Ay. Context modeling and reasoning using ontologies. *University of Technology Berlin*, 2007.

[3] Felix Bachmann, Len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, and Kurt Wallnau. Volume ii: Technical concepts of component-based software engineering. Technical report, Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute, 2000.

[4] Staffan Björk and Jussi Holopainen. Patterns in game design. hingham, ma: Charles river media. *Rolf Kretschmann*, 2005.

[5] Alexander Magno Cordeiro, Glória Maria de Oliveira, Juan Miguel Rentería, and Carlos Alberto Guimarães. Revisão sistemática: uma revisão narrativa. *Rev. Col. Bras. Cir*, 34(6):428–431, 2007.

[6] Lincoln David, Markus Endler, Simone DJ Barbosa, and Jose Viterbo Filho. Middleware support for context-aware mobile applications with adaptive multimodal user interfaces. In *Ubi-Media Computing (U-Media), 2011 4th International Conference on*, pages 106–111. IEEE, 2011.

[7] Regina Borges de Araujo. Computação ubíqua: Princípios, tecnologias e desafios. In *XXI Simpósio Brasileiro de Redes de Computadores*, volume 8, pages 11–13, 2003.

[8] Enrico Del Re, Simone Morosi, Luca Simone Ronga, Sara Jayousi, and Alessio Martinelli. Flexible heterogeneous satellite-based architecture for enhanced quality of life applications. *IEEE Communications Magazine*, 53(5):186–193, 2015.

[9] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166, 2001.

[10] Antonio Di Ferdinando, Alberto Rosi, Ricardo Lent, Antonio Manzalini, and Franco Zambonelli. Myads: A system for adaptive pervasive advertisements. *Pervasive and Mobile computing*, 5(5):385–401, 2009.

[11] Paulo Artur Duarte, Luís Fernando Maia Silva, Francisco Anderson Gomes, Windson Viana, and Fernando Mota Trinta. Dynamic deployment for context-aware multimedia environments. In *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*, pages 197–204. ACM, 2015.

[12] Glauber Ferreira, Emerson Loureiro, Wallace Nogueira, A Gomes, Hyggo Almeida, and A Frery. Uma abordagem baseada em componentes para a construção de edifícios virtuais. In *Proceedings of the 7th Symposium on Virtual Reality*, pages 279–290, 2004.

[13] George H. Forman and John Zahorjan. The challenges of mobile computing. *Comm. ACM*, 36(7):75–84, 1993.

[14] Uwe Hansmann, Lothar Merk, Martin S Nicklous, and Thomas Stober. *Pervasive computing handbook*. Springer Science & Business Media, 2013.

[15] Neal Lathia, Kiran Rachuri, Cecilia Mascolo, and George Roussos. Open source smartphone libraries for computational social science. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 911–920. ACM, 2013.

[16] Jianchao Luo and Hao Feng. A web-based framework for lightweight context-aware mobile applications. *International Journal of Database Theory and Application*, 9(4):119–134, 2016.

[17] Miltiadis D. Lytras. *Ubiquitous and Pervasive Knowledge and Learning Management: Semantics, Social Networking and New Media to Their Full Potential: Semantics, Social Networking and New Media to Their Full Potential*. IGI Global, 2007.

[18] Kalle Lyytinen and Youngjin Yoo. Issues and challenges in ubiquitous computing. *Communications of the ACM*, 45(12):63–96, 2002.

[19] Carsten Magerkurth, Adrian David Cheok, Regan L Mandryk, and Trond Nilsen. Pervasive games: bringing computer entertainment back to the real world. *Computers in Entertainment (CIE)*, 3(3):4–4, 2005.

[20] Marcio E.F. Maia, Andre Fonteles, Benedito Neto, Romulo Gadelha, Windson Viana, and Rossana Andrade. Loccam-loosely coupled context acquisition middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 534–541. ACM, 2013.

[21] Esunly Medina, David Lopez, Roc Meseguer, Sergio F Ochoa, Dolors Royo, and Rodrigo Santos. Mobile autonomous sensing unit (masu): A framework that supports distributed pervasive data sensing. *Sensors*, 16(7):1062, 2016.

[22] Moeiz Miraoui, Chakib Tadj, and C Ben Amar. Architectural survey of context-aware systems in pervasive computing environment. *Ubiquitous Computing and Communication Journal*, 3(3):1–9, 2008.

[23] Roger S. Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[24] Anand Ranganathan and Roy H Campbell. Advertising in a pervasive computing environment. In *Proceedings of the 2nd international workshop on Mobile commerce*, pages 10–14. ACM, 2002.

[25] Lenin Ravindranath, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. Code in the air: simplifying sensing and coordination tasks on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, page 4. ACM, 2012.

[26] Rosana Ferreira Sampaio and Marisa Cotta Mancini. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. *Braz. J. Phys. Ther.(Impr.)*, 11(1):83–89, 2007.

[27] Bill N. Schilit, David M Hilbert, and Jonathan Trevor. Context-aware communication. *IEEE Wireless Communications*, 9(5):46–54, 2002.

[28] Ronny Seiger, Florian Niebling, Mandy Korzetz, Tobias Nicolai, and Thomas Schlegel. A framework for rapid prototyping of multimodal interaction concepts. *Large-scale and Model-based Interactive Systems*, page 21, 2015.

[29] Everton Silva, Larri Botelho, Iverton Santos, and Gustavo Sanchez. Computação ubíqua–definição e exemplos. *Revista de Empreendedorismo, Inovação e Tecnologia*, 2(1):23–32, 2015.

[30] Viviane Gomes Silva, Ranniéry Mazzilly Silva Souza, and Maria João Gomes. Desafios da computação móvel e usabilidade em sala de aula. *Revista de estudios e investigación en psicología y educación*, (13):181–185, 2015.

[31] John Soldatos, Nikolaos Dimakis, Kostas Stamatis, and Lazaros Polymenakos. A breadboard architecture for pervasive context-aware services in smart spaces: middleware components and prototype applications. *Personal and Ubiquitous Computing*, 11(3):193–212, 2007.

[32] Luciana Spagnoli and Karin Becker. Um estudo sobre o desenvolvimento baseado em componentes. *Relatório Técnico do Programa de Pós-Graduação em Ciência da Computação da PUCRS, Porto Alegre, Brasil*, 2003.

[33] Clemens Szyperski, Jan Bosch, and Wolfgang Weck. Component-oriented programming. In *European Conference on Object-Oriented Programming*, pages 184–192. Springer, 1999.

[34] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[35] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[36] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

Apêndice A

Questionários Aplicados após Execução do Experimento

Neste apêndice, serão apresentados todos os questionários que foram aplicados nos questionários realizados para a avaliação deste trabalho de mestrado. Tais questionários, como comentado no capítulo 5, foram respondidos pelos alunos da capacitação do projeto da Sony que participaram do desenvolvimento da aplicação do experimento.

O questionário visou entender dos desenvolvedores qual foi o grau de dificuldade obtido na tarefa de desenvolver a aplicação protótipo que lhes fora pedido. Com isso, buscando comparar se houve uma maior facilidade no desenolvimento quando utilizando a PervasiveSensorLib.

Também extraímos sugestões do que poderia ser alterado tanto na API *Android*, como já na própria biblioteca desenvolvida durante o presente trabalho.

Experimento Sensores - Turma 01

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_1

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	○	◉	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

Função de escutar eventos dos sensores muito genérica, se usa a mesma função para escutar todos eventos dificultando saber de qual sensor pertence o dado que chega, e o formato do dado varia dependendo do sensor

Se você pudesse alterar a API do android no uso de sensores, quais alterações você faria para simplificar o seu uso?

Alteraria a api para que cada sensor tivesse uma função de escuta diferente facilitando a manipulação dos dados

Figura A.1: Turma 1. Questionário 1

Experimento Sensores - Turma 01

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_2

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	○	◉	○	○	Muito Difícil

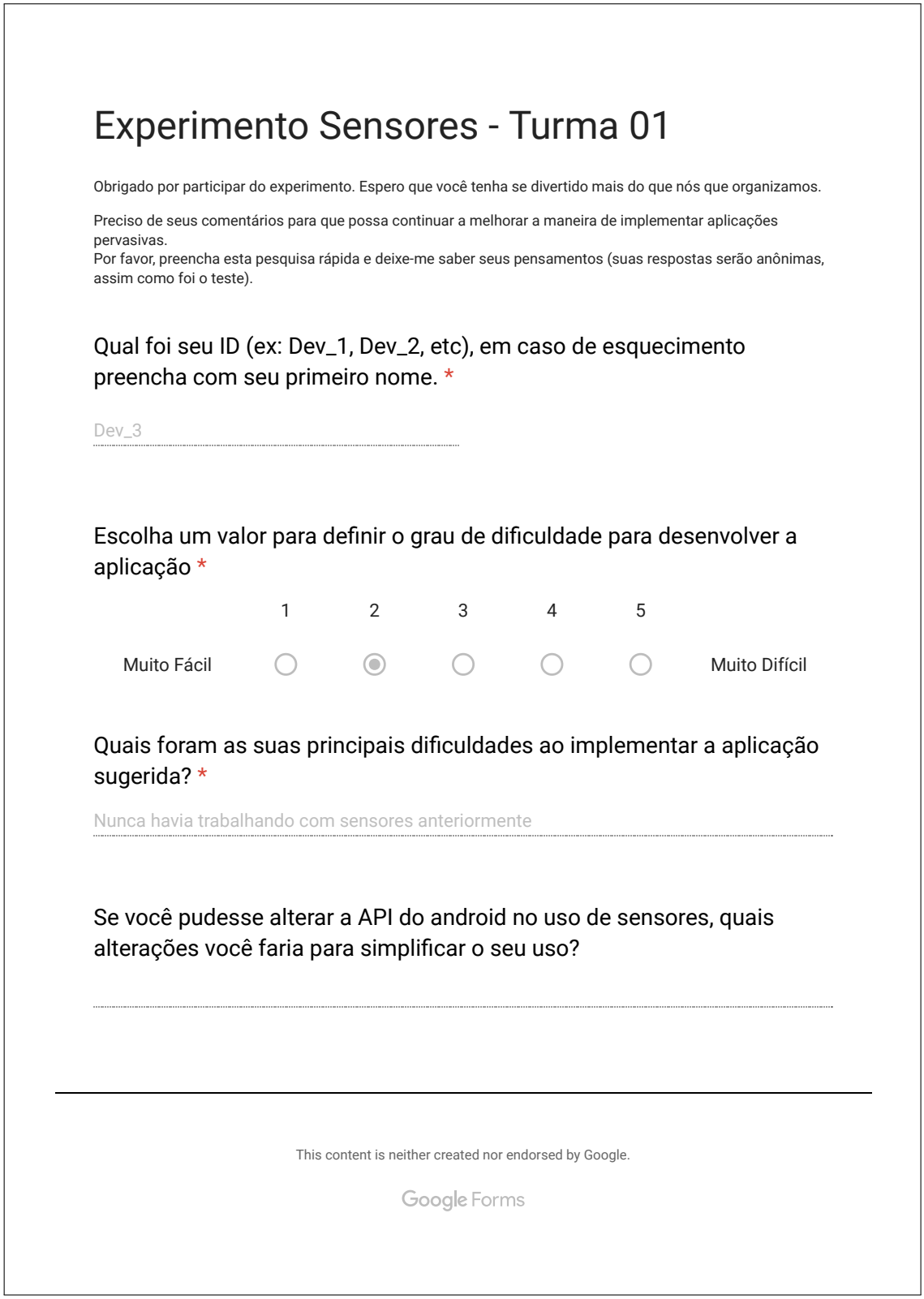
Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

O fato de eu ter focado em como usar os sensores, e não ter prestado atenção a principio quanto a registrar o listener do sensor, acabou atrasando meu tempo de implementação.

Se você pudesse alterar a API do android no uso de sensores, quais alterações você faria para simplificar o seu uso?

Exibir trechos mais completos de código para exemplificar o uso dos sensores, apresentando informações mais diretas de como é feita a implementação das interfaces/classes necessárias para o funcionamento dos mesmos, como por exemplo, a interface Sensor Event Listener. O uso destas é explicado de uma forma detalhada em outras paginas, mas acho que abordar isso de forma breve no contexto de uso dos sensores poderia simplificar e agilizar a pesquisa e assim, a implementação.

Figura A.2: Turma 1. Questionário 2



Experimento Sensores - Turma 01

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_3

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	◉	○	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

Nunca havia trabalhando com sensores anteriormente

Se você pudesse alterar a API do android no uso de sensores, quais alterações você faria para simplificar o seu uso?

This content is neither created nor endorsed by Google.

Google Forms

Figura A.3: Turma 1. Questionário 3

Experimento Sensores - Turma 01

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_4

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	○	○	◉	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

Muita informação de diferentes sensores sendo recebida em um mesmo método. Acaba deixando muito confusa a sua implementação, quando necessitamos utilizar mais de um sensor.

Se você pudesse alterar a API do android no uso de sensores, quais alterações você faria para simplificar o seu uso?

Creio que separar os dados recebidos por cada sensor iría facilitar a implementação, além de deixar o código mais enxuto.

Figura A.4: Turma 1. Questionário 4

Experimento Sensores - Turma 01

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_5

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	○	◉	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

Gerenciar a troca entre os sensores.

Se você pudesse alterar a API do android no uso de sensores, quais alterações você faria para simplificar o seu uso?

Abstrair algumas coisas como por exemplo, o registerListener e unregisterListener, melhorar a captura dos dados de diferentes tipos de sensores, entre outras.

Figura A.5: Turma 1. Questionário 5

Experimento Sensores - Turma 02

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_1

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	◉	○	○	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

Apenas localizar onde fazer as atualizações da UI

Se você pudesse alterar alguns métodos da lib para sensores, quais alterações você faria para simplificar o seu uso? (métodos, parâmetros,...)

Figura A.6: Turma 2. Questionário 1

Experimento Sensores - Turma 02

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_2

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	◉	○	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

A busca pelo local certo de colocar alguma chamada ou fazer alguma alteração.

Se você pudesse alterar alguns métodos da lib para sensores, quais alterações você faria para simplificar o seu uso? (métodos, parâmetros,...)

Percebi que alguns métodos internos para o sensor de orientação estão deprecated, talvez eu tentaria substiruir. ou tentar resolver...

Figura A.7: Turma 2. Questionário 2

Experimento Sensores - Turma 02

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_3

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	○	◉	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

A minha principal dificuldade foi o pouco conhecimento da parte do back-end de android, mas deu para perceber que a aplicação é muito simples de ser implementada com a utilização da biblioteca, caso já exista experiência com o android.

Se você pudesse alterar alguns métodos da lib para sensores, quais alterações você faria para simplificar o seu uso? (métodos, parâmetros,...)

Não encontrei nenhum fator que pudesse ser alterado na lib.

Figura A.8: Turma 2. Questionário 3

Experimento Sensores - Turma 02

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

Dev_4

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	○	◉	○	○	○	Muito Difícil

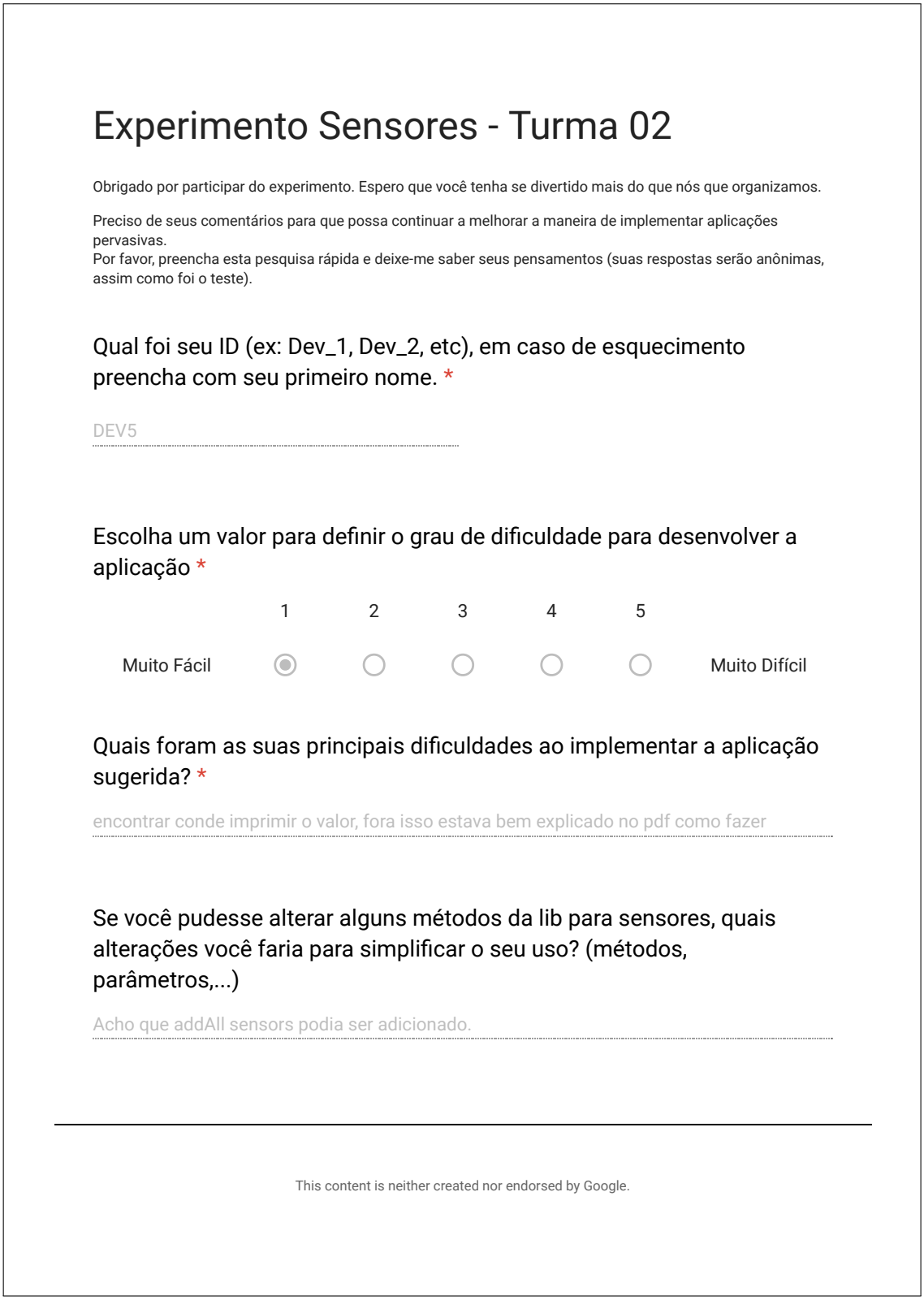
Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

A única dificuldade vivenciada foi em como exibir os dados dos sensores nas fragments de maneira correta.

Se você pudesse alterar alguns métodos da lib para sensores, quais alterações você faria para simplificar o seu uso? (métodos, parâmetros,...)

Sensor.TYPE_ALL poderia ser utilizado como tag para cadastro de todos os sensores e somente na hora de escutar os resultados (onResult callbacks) o desenvolvedor necessitaria definir quais sensores ele quer realmente escutar.

Figura A.9: Turma 2. Questionário 4



Experimento Sensores - Turma 02

Obrigado por participar do experimento. Espero que você tenha se divertido mais do que nós que organizamos.

Preciso de seus comentários para que possa continuar a melhorar a maneira de implementar aplicações pervasivas.
Por favor, preencha esta pesquisa rápida e deixe-me saber seus pensamentos (suas respostas serão anônimas, assim como foi o teste).

Qual foi seu ID (ex: Dev_1, Dev_2, etc), em caso de esquecimento preencha com seu primeiro nome. *

DEV5

Escolha um valor para definir o grau de dificuldade para desenvolver a aplicação *

	1	2	3	4	5	
Muito Fácil	◉	○	○	○	○	Muito Difícil

Quais foram as suas principais dificuldades ao implementar a aplicação sugerida? *

encontrar conde imprimir o valor, fora isso estava bem explicado no pdf como fazer

Se você pudesse alterar alguns métodos da lib para sensores, quais alterações você faria para simplificar o seu uso? (métodos, parâmetros,...)

Acho que addAll sensors podia ser adicionado.

This content is neither created nor endorsed by Google.

Figura A.10: Turma 2. Questionário 5

Apêndice B

Códigos de Exemplo

Código Fonte B.1: Exemplo de código para utilizar mais de um sensor em android

```
1  package accelerometer.android.sensor;
2
3  Imports ***
4
5  public class SensorTestActivity extends Activity implements
       SensorEventListener {
6      private SensorManager sensorManager;
7      private boolean color = false;
8      private View view;
9      private long lastUpdate;
10
11     @Override
12     public void onCreate(Bundle savedInstanceState) {
13         requestWindowFeature(Window.FEATURE_NO_TITLE);
14         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
               WindowManager.LayoutParams.FLAG_FULLSCREEN);
15
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.main);
18         view = findViewById(R.id.textView);
19         view.setBackgroundColor(Color.GREEN);
20
21         sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
22         lastUpdate = System.currentTimeMillis();
```

```
23      }
24
25      @Override
26      public void onSensorChanged(SensorEvent event) {
27          if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
28              float[] values = event.values;
29              float x = values[0];
30              float y = values[1];
31              float z = values[2];
32              float accelationSquareRoot = (x * x + y * y + z * z) / (
                    SensorManager.GRAVITY_EARTH * SensorManager.GRAVITY_EARTH)
                    ;
33              long actualTime = event.timestamp;
34              if (accelationSquareRoot >= 2) {
35                  Toast.makeText(getContext(), "Shaked", Toast.LENGHT_LONG)
                        .show();
36              }
37          } else if(event.sensor.getType() == Sensor.TYPE_LIGHT) {
38              float[] values = event.values;
39              float ilumination = values[0];
40
41              if (ilumination <= 2.0) {
42                Toast.makeText(getContext(), "Dark Place", Toast.
                      LENGHT_LONG).show();
43              }
44          } else if (event.sensor.getType() == Sensor.
                TYPE_AMBIENT_TEMPERATURE) {
45              float[] values = event.values;
46              float amb_temperature = values[0];
47
48              if (amb_temperature <= 18) {
49                Toast.makeText(getContext(), "It is COLD!", Toast.
                      LENGHT_LONG).show();
50              }
51          }
52      }
53
```

```
54      @Override
55      public void onAccuracyChanged(Sensor sensor, int accuracy) {
56
57      }
58
59      @Override
60      protected void onResume() {
61        super.onResume();
62        sensorManager.registerListener(this,
63              sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
64              SensorManager.SENSOR_DELAY_NORMAL);
65      }
66
67      @Override
68      protected void onPause() {
69        super.onPause();
70        sensorManager.unregisterListener(this);
71      }
72  }
```

Apêndice C

Códigos do primeiro experimento

Código Fonte C.1: Código criado pelo desenvolvedor 1.

```
1  package com.example.android.common.fragments;
2
3  import android.content.Context;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.os.Bundle;
9  import android.util.Log;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.view.ViewGroup;
13 import android.widget.ImageView;
14 import android.widget.TextView;
15
16 import com.example.android.slidingtabsbasic.R;
17
18 public class SensorFragment extends android.support.v4.app.Fragment
       implements SensorEventListener {
19
20     private String mTitle;
21     private int mCurDrawable;
22     private TextView mSensorValues;
23     private ImageView mSensorIcon;
```

```
24      private SensorManager mSensorManager;
25      private Sensor mSensor;
26      private int page;
27
28      // newInstance constructor for creating fragment with arguments
29      public static SensorFragment newInstance(int page, String title) {
30          final SensorFragment fragment = new SensorFragment();
31          Bundle args = new Bundle();
32          args.putInt("someInt", page);
33          args.putString("someTitle", title);
34          fragment.setArguments(args);
35          return fragment;
36      }
37
38      public void onResume() {
39          super.onResume();
40          mSensorManager.registerListener(this, mSensor, SensorManager.
                SENSOR_DELAY_NORMAL);
41      }
42
43      public void onPause() {
44          super.onPause();
45          mSensorManager.unregisterListener(this);
46      }
47
48      // Store instance variables based on arguments passed
49      @Override
50      public void onCreate(final Bundle savedInstanceState) {
51          super.onCreate(savedInstanceState);
52          mTitle = getArguments().getString("someTitle");
53          final int curPage = getArguments().getInt("someInt", 0);
54          page = curPage;
55          updateSensorImage(curPage);
56      }
57
58      /**
59       * Register a sensor based on current page.
```

```
60      *
61      * @param page Indicates the tab that is being showed to the user.
62      */
63     private void updateSensorImage(final int page) {
64         switch (page) {
65             case 0:
66                 mCurDrawable = R.drawable.accelerometer_sensor_icon;
67                 Log.e("Tag --------", "To no acc");
68                 break;
69             case 1:
70                 mCurDrawable = R.drawable.light_sensor_icon;
71                 break;
72             case 2:
73                 mCurDrawable = R.drawable.orientation_sensor_field_icon;
74                 break;
75             case 3:
76                 mCurDrawable = R.drawable.proximity_sensor_icon_far;
77                 break;
78         }
79     }
80
81     private void updateSensor(final int page) {
82         switch (page) {
83             case 0:
84                 Log.e("Tag --------", "To no acc");
85                 mSensor = mSensorManager.getDefaultSensor(Sensor.
                       TYPE_ACCELEROMETER);
86                 break;
87             case 1:
88                 mSensor = mSensorManager.getDefaultSensor(Sensor.
                       TYPE_LIGHT);
89                 break;
90             case 2:
91                 mSensor = mSensorManager.getDefaultSensor(Sensor.
                       TYPE_ORIENTATION);
92                 break;
93             case 3:
```

```
94                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_PROXIMITY);
95                  break;
96              }
97          }
98
99          // Inflate the view for the fragment based on layout XML
100         @Override
101         public View onCreateView(final LayoutInflater inflater, final
                ViewGroup container,
102                                  final Bundle savedInstanceState) {
103             final View view = inflater.inflate(R.layout.pager_item, container
                    , false);
104             mSensorIcon = (ImageView) view.findViewById(R.id.sensor_img);
105             mSensorIcon.setImageResource(mCurDrawable);
106             mSensorManager = (SensorManager) getActivity().getSystemService(
                    Context.SENSOR_SERVICE);
107             updateSensor(page);
108             final TextView tvTitle = (TextView) view.findViewById(R.id.
                    item_title);
109             tvTitle.setText(mTitle);
110             mSensorValues = (TextView) view.findViewById(R.id.
                    item_sensor_values);
111
112             //mSensorValues.setText("Sensor Data Goes Here.");
113             return view;
114         }
115
116         @Override
117         public void onDestroyView() {
118             super.onDestroyView();
119         }
120
121         @Override
122         public void onSensorChanged(SensorEvent event) {
123             Log.e("Tag --------", event.values[0] + "");
124             switch (mSensor.getType()) {
```

```
125                 case Sensor.TYPE_ACCELEROMETER:
126                     mSensorValues.setText("X: " + event.values[0] + "\nY: " +
                            event.values[1] + "\nZ: " + event.values[2]);
127                     break;
128                 case Sensor.TYPE_LIGHT:
129                     mSensorValues.setText(event.values[0] + " Sl lux units");
130                     if (event.values[0] < 2) {
131                         mSensorIcon.setImageDrawable(getActivity().
                                getDrawable(R.drawable.light_sensor_icon_off));
132                     } else {
133                         mSensorIcon.setImageDrawable(getActivity().
                                getDrawable(R.drawable.light_sensor_icon));
134                     }
135                     break;
136                 case Sensor.TYPE_ORIENTATION:
137                     mSensorIcon.setRotation(event.values[0]);
138                     break;
139                 case Sensor.TYPE_PROXIMITY:
140                     if (event.values[0] == 0) {
141                         mSensorIcon.setImageDrawable(getActivity().
                                getDrawable(R.drawable.proximity_sensor_icon));
142                         mSensorValues.setText("Perto");
143                     } else {
144                         mSensorValues.setText("Longe");
145                         mSensorIcon.setImageDrawable(getActivity().
                                getDrawable(R.drawable.proximity_sensor_icon_far))
                                ;
146                     }
147                     break;
148             }
149         }
150
151         @Override
152         public void onAccuracyChanged(Sensor sensor, int accuracy) {
153
154         }
155 }
```

Código Fonte C.2: Código criado pelo desenvolvedor 2.

```
1  package com.example.android.common.fragments;
2
3  import android.content.Context;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.os.Bundle;
9  import android.util.Log;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.view.ViewGroup;
13 import android.widget.ImageView;
14 import android.widget.TextView;
15
16 import com.example.android.slidingtabsbasic.R;
17
18 public class SensorFragment extends android.support.v4.app.Fragment
       implements SensorEventListener{
19
20     private String mTitle;
21     private int mCurDrawable;
22     private TextView mSensorValues;
23     private ImageView mSensorIcon;
24     private Sensor mSensor;
25     private SensorManager mSensorManager;
26
27     // newInstance constructor for creating fragment with arguments
28     public static SensorFragment newInstance(int page, String title) {
29         final SensorFragment fragment = new SensorFragment();
30         Bundle args = new Bundle();
31         args.putInt("someInt", page);
32         args.putString("someTitle", title);
33         fragment.setArguments(args);
34         return fragment;
35     }
```

```
36
37      // Store instance variables based on arguments passed
38      @Override
39      public void onCreate(final Bundle savedInstanceState) {
40          super.onCreate(savedInstanceState);
41          mTitle = getArguments().getString("someTitle");
42          final int curPage = getArguments().getInt("someInt", 0);
43
44          mSensorManager = (SensorManager) getActivity().getSystemService(
                Context.SENSOR_SERVICE);
45
46          updateSensorImage(curPage);
47
48      }
49
50      /**
51       * Register a sensor based on current page.
52       * @param page Indicates the tab that is being showed to the user.
53       */
54      private void updateSensorImage(final int page) {
55          switch (page) {
56              case 0:
57                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
58                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_ACCELEROMETER);
59                  break;
60              case 1:
61                  mCurDrawable = R.drawable.light_sensor_icon;
62                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_LIGHT);
63                  break;
64              case 2:
65                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
66                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_ORIENTATION);
67                  break;
68              case 3:
```

```
69                  mCurDrawable = R.drawable.proximity_sensor_icon_far;
70                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_PROXIMITY);
71                  break;
72              }
73          }
74
75          // Inflate the view for the fragment based on layout XML
76          @Override
77          public View onCreateView(final LayoutInflater inflater, final
                ViewGroup container,
78                                   final Bundle savedInstanceState) {
79              final View view = inflater.inflate(R.layout.pager_item, container
                    , false);
80              mSensorIcon = (ImageView) view.findViewById(R.id.sensor_img);
81              mSensorIcon.setImageResource(mCurDrawable);
82
83              final TextView tvTitle = (TextView) view.findViewById(R.id.
                    item_title);
84              tvTitle.setText(mTitle);
85              mSensorValues = (TextView) view.findViewById(R.id.
                    item_sensor_values);
86
87              return view;
88          }
89
90          @Override
91          public void onDestroyView() {
92              super.onDestroyView();
93          }
94
95          @Override
96          public void onSensorChanged(SensorEvent event) {
97              switch (mSensor.getType()) {
98                  case Sensor.TYPE_ACCELEROMETER:
99                      mSensorValues.setText(event.values[0] + "\n" + event.
                            values[1] + "\n" + event.values[2]);
```

```
100                 break;
101             case Sensor.TYPE_LIGHT:
102
103                 if(event.values[0] < 2){
104                     mSensorIcon.setImageDrawable(getActivity().
                            getDrawable(R.drawable.light_sensor_icon_off));
105                 }
106                 else{
107                     mSensorIcon.setImageDrawable(getActivity().
                            getDrawable(R.drawable.light_sensor_icon));
108                 }
109                 break;
110             case Sensor.TYPE_ORIENTATION:
111                 mSensorIcon.setRotation(event.values[0]);
112                 break;
113             case Sensor.TYPE_PROXIMITY:
114                 if(event.values[0] < 1){
115                     mSensorIcon.setImageDrawable(getActivity().
                            getDrawable(R.drawable.proximity_sensor_icon));
116                 }
117                 else{
118                     mSensorIcon.setImageDrawable(getActivity().
                            getDrawable(R.drawable.proximity_sensor_icon_far))
                            ;
119                 }
120                 break;
121         }
122
123     }
124
125     @Override
126     public void onResume() {
127         // Register a listener for the sensor.
128         super.onResume();
129         mSensorManager.registerListener(this, mSensor, SensorManager.
                SENSOR_DELAY_NORMAL);
130     }
```

```
131
132     @Override
133     public void onAccuracyChanged(Sensor sensor, int accuracy) {
134
135     }
136
137     @Override
138     public void onPause() {
139         // Be sure to unregister the sensor when the activity pauses.
140         super.onPause();
141         mSensorManager.unregisterListener(this);
142     }
143 }
```

Código Fonte C.3: Código criado pelo desenvolvedor 3.

```
1  package com.example.android.common.fragments;
2
3  import android.content.Context;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.os.Bundle;
9  import android.util.Log;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.view.ViewGroup;
13 import android.widget.ImageView;
14 import android.widget.TextView;
15
16 import com.example.android.slidingtabsbasic.R;
17
18 public class SensorFragment extends android.support.v4.app.Fragment
       implements SensorEventListener {
19
20     private String mTitle;
21     private int mCurDrawable;
```

```
22      private TextView mSensorValues;
23      private ImageView mSensorIcon;
24      private SensorManager mSensorManager;
25      private Sensor mAccel;
26      private Sensor mLight;
27      private Sensor mOrientation;
28      private Sensor mProximity;
29      private double[] gravity = new double[]{0, 0, 0};
30      private double[] linear_acceleration = new double[]{0, 0, 0};
31      private double[] orientation = new double[]{0, 0, 0};
32      private double light = 0;
33      private double distance = 0;
34      private int currentTab = 0;
35
36
37      // newInstance constructor for creating fragment with arguments
38      public static SensorFragment newInstance(int page, String title) {
39
40          final SensorFragment fragment = new SensorFragment();
41          Bundle args = new Bundle();
42          args.putInt("someInt", page);
43          args.putString("someTitle", title);
44          fragment.setArguments(args);
45          return fragment;
46      }
47
48      // Store instance variables based on arguments passed
49      @Override
50      public void onCreate(final Bundle savedInstanceState) {
51          super.onCreate(savedInstanceState);
52
53          mTitle = getArguments().getString("someTitle");
54          final int curPage = getArguments().getInt("someInt", 0);
55          mSensorManager = (SensorManager) this.getContext().
                getSystemService(Context.SENSOR_SERVICE);
56          mAccel = mSensorManager.getDefaultSensor(Sensor.
                TYPE_ACCELEROMETER);
```

```
57          mProximity = mSensorManager.getDefaultSensor(Sensor.
                TYPE_PROXIMITY);
58          mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
59          mOrientation = mSensorManager.getDefaultSensor(Sensor.
                TYPE_ORIENTATION);
60
61          updateSensorImage(curPage);
62
63      }
64
65      /**
66       * Register a sensor based on current page.
67       *
68       * @param page Indicates the tab that is being showed to the user.
69       */
70      private void updateSensorImage(final int page) {
71          currentTab = page;
72          switch (page) {
73              case 0:
74                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
75                  break;
76              case 1:
77                  if (light > 5) {
78                      mCurDrawable = R.drawable.light_sensor_icon;
79                  } else {
80                      mCurDrawable = R.drawable.light_sensor_icon_off;
81                  }
82                  break;
83              case 2:
84                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
85                  break;
86              case 3:
87                  if (distance > 0.5)
88                      mCurDrawable = R.drawable.proximity_sensor_icon_far;
89                  else
90                      mCurDrawable = R.drawable.proximity_sensor_icon;
91                  break;
```

```
92          }
93      }
94
95      // Inflate the view for the fragment based on layout XML
96      @Override
97      public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
98                               final Bundle savedInstanceState) {
99          final View view = inflater.inflate(R.layout.pager_item, container
                , false);
100         mSensorIcon = (ImageView) view.findViewById(R.id.sensor_img);
101         mSensorIcon.setImageResource(mCurDrawable);
102
103         final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
104         tvTitle.setText(mTitle);
105         mSensorValues = (TextView) view.findViewById(R.id.
                item_sensor_values);
106         mSensorValues.setText("Sensor Data Goes Here.");
107         mSensorManager.registerListener(this, mAccel, SensorManager.
                SENSOR_DELAY_UI);
108         mSensorManager.registerListener(this, mProximity, SensorManager.
                SENSOR_DELAY_UI);
109         mSensorManager.registerListener(this, mLight, SensorManager.
                SENSOR_DELAY_UI);
110         mSensorManager.registerListener(this, mOrientation, SensorManager
                .SENSOR_DELAY_UI);
111
112
113         return view;
114     }
115
116
117     public void onSensorChanged(SensorEvent event) {
118
119
120         if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
```

```
121             Log.d("Check", "Acelerometro");
122
123             final double alpha = 0.8;
124
125             gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values
                    [0];
126             gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values
                    [1];
127             gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values
                    [2];
128
129             // Remove the gravity contribution with the high-pass filter.
130             linear_acceleration[0] = event.values[0] - gravity[0];
131             linear_acceleration[1] = event.values[1] - gravity[1];
132             linear_acceleration[2] = event.values[2] - gravity[2];
133
134
135         } else if (event.sensor.getType() == Sensor.TYPE_PROXIMITY) {
136             Log.d("Check", "Proximidade");
137             distance = event.values[0];
138         } else if (event.sensor.getType() == Sensor.TYPE_LIGHT) {
139             Log.d("Check", "Light");
140             light = event.values[0];
141         } else if (event.sensor.getType() == Sensor.TYPE_ORIENTATION) {
142             Log.d("Check", "Magnetic");
143             orientation[0] = event.values[0];
144             orientation[1] = event.values[1];
145             orientation[2] = event.values[2];
146
147         }
148
149         if (currentTab == 0) {
150             mSensorValues.setText("Acelerometro\nX=" +
151                     String.format(" %1$.4f", gravity[0]) + "...\nY=" +
152                     String.format(" %1$.4f", gravity[1]) + "...\nZ=" +
153                     String.format(" %1$.4f", gravity[2]) + "...");
154         } else if (currentTab == 1) {
```

```
155                 if (light > 5) {
156                     mCurDrawable = R.drawable.light_sensor_icon;
157                 } else {
158                     mCurDrawable = R.drawable.light_sensor_icon_off;
159                 }
160             } else if (currentTab == 2) {
161                 mSensorIcon.setRotation((float) orientation[0]);
162                 mSensorValues.setText("");
163             } else if (currentTab == 3) {
164                 if (distance < 0.5) {
165                     mSensorValues.setText("Near");
166                     mCurDrawable = R.drawable.proximity_sensor_icon;
167                 } else {
168                     mSensorValues.setText("");
169                     mCurDrawable = R.drawable.proximity_sensor_icon_far;
170                 }
171             }
172         }
173 
174         @Override
175         public void onAccuracyChanged(Sensor sensor, int accuracy) {
176             //Nothing
177         }
178 
179         @Override
180         public void onDestroyView() {
181             super.onDestroyView();
182         }
183 
184 }
```

Código Fonte C.4: Código criado pelo desenvolvedor 4.

```
1  package com.example.android.common.fragments;
2 
3  import android.content.Context;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
```

```
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.hardware.TriggerEvent;
9  import android.hardware.TriggerEventListener;
10 import android.os.Bundle;
11 import android.util.Log;
12 import android.view.LayoutInflater;
13 import android.view.View;
14 import android.view.ViewGroup;
15 import android.widget.ImageView;
16 import android.widget.TextView;
17
18 import com.example.android.slidingtabsbasic.R;
19
20 public class SensorFragment extends android.support.v4.app.Fragment
       implements SensorEventListener {
21
22     private static SensorManager mSensorManager;
23     private String mTitle;
24     private int mCurDrawable;
25     private TextView mSensorValues;
26     private ImageView mSensorIcon;
27     private Sensor mSensor;
28     private double[] gravity;
29     private double[] linear_acceleration;
30     private String data;
31
32     // newInstance constructor for creating fragment with arguments
33     public static SensorFragment newInstance(int page, String title) {
34
35         final SensorFragment fragment = new SensorFragment();
36         Bundle args = new Bundle();
37         args.putInt("someInt", page);
38         args.putString("someTitle", title);
39         fragment.setArguments(args);
40         return fragment;
41     }
```

```
42
43      // Store instance variables based on arguments passed
44      @Override
45      public void onCreate(final Bundle savedInstanceState) {
46          super.onCreate(savedInstanceState);
47          mTitle = getArguments().getString("someTitle");
48          final int curPage = getArguments().getInt("someInt", 0);
49          updateSensorImage(curPage);
50          Context ctx = getContext();
51          mSensorManager = (SensorManager) ctx.getSystemService(Context.
                SENSOR_SERVICE);
52
53          if (curPage == 0) {
54              mSensor = mSensorManager.getDefaultSensor(Sensor.
                    TYPE_ACCELEROMETER);
55
56          } else if (curPage == 1) {
57              mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
58
59          } else if (curPage == 2) {
60              mSensor = mSensorManager.getDefaultSensor(Sensor.
                    TYPE_ORIENTATION);
61
62          } else {
63              mSensor = mSensorManager.getDefaultSensor(Sensor.
                    TYPE_PROXIMITY);
64
65          }
66
67      }
68
69      @Override
70      public final void onSensorChanged(SensorEvent event) {
71
72          if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
73              data = "x= " + Float.toString(event.values[0])  + "\n" + "y =
                        " + Float.toString(event.values[1]) + "\n " + "z = " +
```

```
                Float.toString(event.values[2]) ;
74
75          }else if(event.sensor.getType() == Sensor.TYPE_LIGHT){
76              if(event.values[0] < 5.0f){
77                  mSensorIcon.setImageDrawable(getActivity().getDrawable(R.
                        drawable.light_sensor_icon_off));
78              }else{
79                  mSensorIcon.setImageDrawable(getActivity().getDrawable(R.
                        drawable.light_sensor_icon));
80              }
81
82          }else if(event.sensor.getType() == Sensor.TYPE_ORIENTATION){
83              mSensorIcon.setRotation(event.values[0]);
84
85          }else if(event.sensor.getType() == Sensor.TYPE_PROXIMITY){
86              if(event.values[0] == 0) {
87                  mSensorIcon.setImageDrawable(getActivity().getDrawable(R.
                        drawable.proximity_sensor_icon));
88
89
90              }else{
91                  mSensorIcon.setImageDrawable(getActivity().getDrawable(R.
                        drawable.proximity_sensor_icon_far));
92
93              }
94          }
95          mSensorValues.setText(data);
96
97      }
98
99      @Override
100     public void onAccuracyChanged(Sensor sensor, int accuracy) {
101
102     }
103
104     /**
105      * Register a sensor based on current page.
```

```
106         *
107         * @param page Indicates the tab that is being showed to the user.
108         */
109        private void updateSensorImage(final int page) {
110            switch (page) {
111                case 0:
112                    mCurDrawable = R.drawable.accelerometer_sensor_icon;
113                    break;
114                case 1:
115                    mCurDrawable = R.drawable.light_sensor_icon;
116                    break;
117                case 2:
118                    mCurDrawable = R.drawable.orientation_sensor_field_icon;
119                    break;
120                case 3:
121                    mCurDrawable = R.drawable.proximity_sensor_icon_far;
122                    break;
123            }
124        }
125
126        // Inflate the view for the fragment based on layout XML
127        @Override
128        public View onCreateView(final LayoutInflater inflater, final
               ViewGroup container,
129                                 final Bundle savedInstanceState) {
130            final View view = inflater.inflate(R.layout.pager_item, container
                   , false);
131            mSensorIcon = (ImageView) view.findViewById(R.id.sensor_img);
132            mSensorIcon.setImageResource(mCurDrawable);
133            mSensorValues = (TextView) view.findViewById(R.id.
                   item_sensor_values);
134
135            final TextView tvTitle = (TextView) view.findViewById(R.id.
                   item_title);
136            tvTitle.setText(mTitle);
137
138            return view;
```

```
139         }
140
141         @Override
142         public void onResume() {
143             super.onResume();
144             mSensorManager.registerListener(this, mSensor, SensorManager.
                    SENSOR_DELAY_NORMAL);
145         }
146
147         @Override
148         public void onPause() {
149             super.onPause();
150             mSensorManager.unregisterListener(this);
151         }
152
153
154         @Override
155         public void onDestroyView() {
156             super.onDestroyView();
157         }
158 }
```

Código Fonte C.5: Código criado pelo desenvolvedor 5.

```
1  package com.example.android.common.fragments;
2
3  import android.content.Context;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.os.Bundle;
9  import android.view.LayoutInflater;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.widget.ImageView;
13 import android.widget.TextView;
14 import android.widget.Toast;
```

```
15
16 import com.example.android.slidingtabsbasic.R;
17
18 public class SensorFragment extends android.support.v4.app.Fragment
       implements SensorEventListener {
19
20
21     private String mTitle;
22     private int mCurDrawable;
23     private TextView mSensorValues;
24     private ImageView mSensorIcon;
25
26     private SensorManager mSensorManager;
27
28     private Sensor mSensor;
29
30     private int curPage;
31
32     private float x, y, z;
33     private Sensor mProximitySensor, mAccelerometerSensor, mLigth;
34
35
36     // newInstance constructor for creating fragment with arguments
37     public static SensorFragment newInstance(int page, String title) {
38         final SensorFragment fragment = new SensorFragment();
39         Bundle args = new Bundle();
40         args.putInt("someInt", page);
41         args.putString("someTitle", title);
42         fragment.setArguments(args);
43
44         return fragment;
45     }
46
47     // Store instance variables based on arguments passed
48     @Override
49     public void onCreate(final Bundle savedInstanceState) {
50         super.onCreate(savedInstanceState);
```

```
51          mTitle = getArguments().getString("someTitle");
52          curPage = getArguments().getInt("someInt", 0);
53          updateSensorImage(curPage);
54
55          mSensorManager = (SensorManager) getActivity().getSystemService(
                Context.SENSOR_SERVICE);
56
57          switch (curPage) {
58              case 0:
59                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_ACCELEROMETER);
60                  mSensorManager.registerListener(this, mSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
61                  break;
62              case 1:
63                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_LIGHT);
64                  mSensorManager.registerListener(this, mSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
65              case 2:
66                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_ORIENTATION);
67                  mSensorManager.registerListener(this, mSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
68              case 3:
69                  mSensor = mSensorManager.getDefaultSensor(Sensor.
                        TYPE_PROXIMITY);
70                  mSensorManager.registerListener(this, mSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
71                  break;
72          }
73      }
74
75      @Override
76      public final void onSensorChanged(SensorEvent event) {
77          if (event == null) {
78              return;
```

```
79          }
80
81          x = event.values[0];
82          y = event.values[1];
83          z = event.values[2];
84
85          if(curPage == 0){
86              mSensorValues.setText("x " + z + " y " + y + " z " + z);
87          }else if(curPage == 1){
88              mSensorValues.setText(x + "");
89              if(x == 0) {
90                  mSensorIcon.setImageResource(R.drawable.
                        light_sensor_icon_off);
91              }else{
92                  mSensorIcon.setImageResource(R.drawable.light_sensor_icon
                        );
93              }
94          }else if(curPage == 2){
95              mSensorValues.setText(x + "");
96              mSensorIcon.setRotation(x);
97          }else {
98              mSensorValues.setText(x + "");
99              if(x == 0) {
100                 mSensorIcon.setImageResource(R.drawable.
                        proximity_sensor_icon);
101             }else{
102                 mSensorIcon.setImageResource(R.drawable.
                        proximity_sensor_icon_far);
103             }
104         }
105     }
106
107     @Override
108     public void onAccuracyChanged(Sensor sensor, int i) {
109
110     }
111
```

```
112     /**
113      * Register a sensor based on current page.
114      *
115      * @param page Indicates the tab that is being showed to the user.
116      */
117     private void updateSensorImage(final int page) {
118
119         switch (page) {
120             case 0:
121                 mCurDrawable = R.drawable.accelerometer_sensor_icon;
122
123                 break;
124             case 1:
125                 mCurDrawable = R.drawable.light_sensor_icon;
126
127                 break;
128             case 2:
129                 mCurDrawable = R.drawable.orientation_sensor_field_icon;
130
131                 break;
132             case 3:
133                 mCurDrawable = R.drawable.proximity_sensor_icon_far;
134
135                 break;
136         }
137     }
138
139     // Inflate the view for the fragment based on layout XML
140     @Override
141     public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
142                              final Bundle savedInstanceState) {
143         final View view = inflater.inflate(R.layout.pager_item, container
                , false);
144         mSensorIcon = (ImageView) view.findViewById(R.id.sensor_img);
145         mSensorIcon.setImageResource(mCurDrawable);
146
```

```
147             final TextView tvTitle = (TextView) view.findViewById(R.id.
                    item_title);
148             tvTitle.setText(mTitle);
149
150             mSensorValues = (TextView) view.findViewById(R.id.
                    item_sensor_values);
151             mSensorValues.setText(Float.toString(x));
152
153             return view;
154         }
155
156         @Override
157         public void onDestroyView() {
158             super.onDestroyView();
159         }
160
161     }
```

Apêndice D

Códigos do segundo experimento

Código Fonte D.1: Código criado pelo desenvolvedor 1.

```
1  package com.example.android.common.fragments;
2
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.View;
8  import android.view.ViewGroup;
9  import android.widget.ImageView;
10 import android.widget.TextView;
11 import com.example.android.slidingtabsbasic.R;
12 import development.master.com.pervasivesensorlib.PervasiveSensorManager;
13 import development.master.com.pervasivesensorlib.SensorCallback;
14 import development.master.com.pervasivesensorlib.SensorCallbackAdapter;
15
16 public class SensorFragment extends android.support.v4.app.Fragment {
17
18     private String mTitle;
19     private int mCurDrawable;
20     private TextView sensorValue;
21     private int tab;
22
23     private PervasiveSensorManager mSensorManager;
24
```

```
25      // newInstance constructor for creating fragment with arguments
26      public static SensorFragment newInstance(final int page, final String
            title) {
27          final SensorFragment fragment = new SensorFragment();
28          Bundle args = new Bundle();
29          args.putInt("someInt", page);
30          args.putString("someTitle", title);
31          fragment.setArguments(args);
32          return fragment;
33      }
34
35      // Store instance variables based on arguments passed
36      @Override
37      public void onCreate(final Bundle savedInstanceState) {
38          super.onCreate(savedInstanceState);
39          final int curPage = getArguments().getInt("someInt", 0);
40          mTitle = getArguments().getString("someTitle");
41          registerSensorListener(curPage);
42
43          mSensorManager = new PervasiveSensorManager(getContext());
44          mSensorManager.addSensor(Sensor.TYPE_ACCELEROMETER);
45          mSensorManager.addSensor(Sensor.TYPE_LIGHT);
46          mSensorManager.addSensor(Sensor.TYPE_GYROSCOPE);
47          mSensorManager.addSensor(Sensor.TYPE_PROXIMITY);
48          mSensorManager.setSensorsCallback(createSensorCallbacks());
49          mSensorManager.setUpdateTime(Sensor.TYPE_ACCELEROMETER,
                SensorManager.SENSOR_DELAY_FASTEST);
50      }
51
52      /**
53       * Register a sensor based on current page.
54       * @param page Indicates the tab that is being showed to the user.
55       */
56      private void registerSensorListener(final int page) {
57          tab = page;
58          switch (page) {
59              case 0:
```

```
60                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
61                  break;
62              case 1:
63                  mCurDrawable = R.drawable.light_sensor_icon;
64                  break;
65              case 2:
66                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
67                  break;
68              case 3:
69                  mCurDrawable = R.drawable.proximity_sensor_icon;
70                  break;
71          }
72      }
73
74      // Inflate the view for the fragment based on layout XML
75      @Override
76      public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
77                               final Bundle savedInstanceState) {
78          final View view = inflater.inflate(R.layout.pager_item, container
                , false);
79          final ImageView ivIcon = (ImageView) view.findViewById(R.id.
                sensor_img);
80          ivIcon.setImageResource(mCurDrawable);
81          final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
82          tvTitle.setText(mTitle);
83          sensorValue = (TextView) view.findViewById(R.id.
                item_sensor_values);
84          return view;
85      }
86
87      @Override
88      public void onDestroyView() {
89          super.onDestroyView();
90      }
91
```

```
92      private SensorCallback createSensorCallbacks(){
93          return new SensorCallbackAdapter() {
94              @Override
95              public void onReceiveAccelerometerData(float x, float y,
                    float z) {
96                  super.onReceiveAccelerometerData(x, y, z);
97                  if (tab == 0){
98                      sensorValue.setText(Float.toString(x) + '\n' + Float.
                            toString(y) + '\n' + Float.toString(z));
99                  }
100             }
101
102             @Override
103             public void onReceiveLightData(float light) {
104                 super.onReceiveLightData(light);
105                 if (tab == 1){
106                     sensorValue.setText(Float.toString(light));
107                 }
108             }
109
110             @Override
111             public void onReceiveGyroscopeData(float x, float y, float z)
                    {
112                 super.onReceiveGyroscopeData(x, y, z);
113                 if (tab == 2){
114                     sensorValue.setText(Float.toString(x) + '\n' + Float.
                            toString(y) + '\n' + Float.toString(z));
115                 }
116
117             }
118
119             @Override
120             public void onReceiveProximityData(boolean isNear) {
121                 super.onReceiveProximityData(isNear);
122                 if (tab == 3){
123                     if (isNear){
124                         sensorValue.setText("NEAR");
```

```
125                         } else {
126                             sensorValue.setText("FAR");
127                         }
128                     }
129                 }
130             };
131     }
132 }
```

Código Fonte D.2: Código criado pelo desenvolvedor 2.

```
1  package com.example.android.common.fragments;
2
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.view.LayoutInflater;
8  import android.view.View;
9  import android.view.ViewGroup;
10 import android.widget.ImageView;
11 import android.widget.TextView;
12 import com.example.android.slidingtabsbasic.R;
13 import development.master.com.pervasivesensorlib.PervasiveSensorManager;
14 import development.master.com.pervasivesensorlib.SensorCallback;
15 import development.master.com.pervasivesensorlib.SensorCallbackAdapter;
16
17 public class SensorFragment extends android.support.v4.app.Fragment {
18
19     private String mTitle;
20     private TextView mValue;
21     private int mCurDrawable;
22     private ImageView ivIcon;
23     private int mPage;
24
25     private PervasiveSensorManager sensorManager;
26
27     // newInstance constructor for creating fragment with arguments
```

```
28      public static SensorFragment newInstance(final int page, final String
            title) {
29          final SensorFragment fragment = new SensorFragment();
30          Bundle args = new Bundle();
31          args.putInt("someInt", page);
32          args.putString("someTitle", title);
33          fragment.setArguments(args);
34          return fragment;
35      }
36
37      // Store instance variables based on arguments passed
38      @Override
39      public void onCreate(final Bundle savedInstanceState) {
40          super.onCreate(savedInstanceState);
41          final int curPage = getArguments().getInt("someInt", 0);
42          mTitle = getArguments().getString("someTitle");
43          registerSensorListener(curPage);
44          sensorManager = new PervasiveSensorManager(getContext());
45          sensorManager.addSensor(Sensor.TYPE_ACCELEROMETER);
46          sensorManager.addSensor(Sensor.TYPE_LIGHT);
47          sensorManager.addSensor(Sensor.TYPE_ORIENTATION);
48          sensorManager.addSensor(Sensor.TYPE_PROXIMITY);
49          sensorManager.setSensorsCallback(createSensorCallbacks());
50
51          sensorSupported();
52          sensorUpdateTime();
53      }
54
55      /**
56       * Register a sensor based on current page.
57       *
58       * @param page Indicates the tab that is being showed to the user.
59       */
60      private void registerSensorListener(int page) {
61          mPage = page;
62          switch (page) {
63              case 0:
```

```
64                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
65                  break;
66              case 1:
67                  mCurDrawable = R.drawable.light_sensor_icon;
68                  break;
69              case 2:
70                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
71                  break;
72              case 3:
73                  mCurDrawable = R.drawable.proximity_sensor_icon;
74                  break;
75          }
76      }
77
78      // Inflate the view for the fragment based on layout XML
79      @Override
80      public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
81                               final Bundle savedInstanceState) {
82          final View view = inflater.inflate(R.layout.pager_item, container
                , false);
83          ivIcon = (ImageView) view.findViewById(R.id.sensor_img);
84          ivIcon.setImageResource(mCurDrawable);
85          final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
86          tvTitle.setText(mTitle);
87          mValue = (TextView) view.findViewById(R.id.item_sensor_values);
88          return view;
89      }
90
91      @Override
92      public void onDestroyView() {
93          super.onDestroyView();
94      }
95
96      private SensorCallback createSensorCallbacks() {
97          return new SensorCallbackAdapter() {
```

```
98          @Override
99          public void onReceiveAccelerometerData(float v, float v1,
                float v2) {
100             if (mPage == 0) {
101                 super.onReceiveAccelerometerData(v, v1, v2);
102                 mValue.setText("x: " + v + "\ny: " + v1 + "\nz: " +
                        v2);
103             }
104         }
105
106         @Override
107         public void onReceiveLightData(float v) {
108             if (mPage == 1) {
109                 super.onReceiveLightData(v);
110                 mValue.setText("x < " + v);
111             }
112         }
113
114         @Override
115         public void onReceiveOrientationData(float v) {
116             if (mPage == 2) {
117                 super.onReceiveOrientationData(v);
118                 ivIcon.setRotation(v);
119             }
120         }
121
122         @Override
123         public void onReceiveProximityData(boolean b) {
124             super.onReceiveProximityData(b);
125             if (mPage == 3) {
126                 if (b) {
127                     ivIcon.setImageResource(R.drawable.
                            proximity_sensor_icon);
128                     mValue.setText("Near");
129                 } else {
130                     ivIcon.setImageResource(R.drawable.
                            proximity_sensor_icon_far);
```

```
131                     }
132                 }
133             }
134         };
135     }
136
137     public void sensorSupported(){
138         if (!sensorManager.isSupported(Sensor.TYPE_ACCELEROMETER)){
139             Log.e("Sensor Support", "Type_Accelerometer is not supported
                    on this device");
140         }
141         if (!sensorManager.isSupported(Sensor.TYPE_LIGHT)){
142             Log.e("Sensor Support", "Type_Light is not supported on this
                    device");
143         }
144         if (!sensorManager.isSupported(Sensor.TYPE_ORIENTATION)){
145             Log.e("Sensor Support", "Type_Orientation is not supported on
                     this device");
146         }
147         if (!sensorManager.isSupported(Sensor.TYPE_PROXIMITY)){
148             Log.e("Sensor Support", "Type_Proximity is not supported on
                    this device");
149         }
150     }
151
152     public void sensorUpdateTime() {
153         sensorManager.setUpdateTime(Sensor.TYPE_ACCELEROMETER,
                SensorManager.SENSOR_DELAY_FASTEST);
154         sensorManager.setUpdateTime(Sensor.TYPE_LIGHT, SensorManager.
                SENSOR_DELAY_FASTEST);
155         sensorManager.setUpdateTime(Sensor.TYPE_ORIENTATION,
                SensorManager.SENSOR_DELAY_FASTEST);
156         sensorManager.setUpdateTime(Sensor.TYPE_PROXIMITY, SensorManager.
                SENSOR_DELAY_FASTEST);
157     }
158 }
```

Código Fonte D.3: Código criado pelo desenvolvedor 3.

```
1  package com.example.android.common.fragments;
2
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.View;
8  import android.view.ViewGroup;
9  import android.widget.ImageView;
10 import android.widget.TextView;
11 import com.example.android.slidingtabsbasic.R;
12 import development.master.com.pervasivesensorlib.PervasiveSensorManager;
13 import development.master.com.pervasivesensorlib.SensorCallback;
14 import development.master.com.pervasivesensorlib.SensorCallbackAdapter;
15
16 public class SensorFragment extends android.support.v4.app.Fragment {
17
18     private String mTitle;
19     private int mCurDrawable;
20     private PervasiveSensorManager mSensorManager;
21     private TextView sensorValues;
22     private ImageView ivIcon;
23     private int mPage;
24
25     // newInstance constructor for creating fragment with arguments
26     public static SensorFragment newInstance(final int page, final String
           title) {
27         final SensorFragment fragment = new SensorFragment();
28         Bundle args = new Bundle();
29         args.putInt("someInt", page);
30         args.putString("someTitle", title);
31         fragment.setArguments(args);
32         return fragment;
33     }
34
35     // Store instance variables based on arguments passed
```

```
36      @Override
37      public void onCreate(final Bundle savedInstanceState) {
38          super.onCreate(savedInstanceState);
39          final int curPage = getArguments().getInt("someInt", 0);
40          mTitle = getArguments().getString("someTitle");
41          registerSensorListener(curPage);
42          mSensorManager = new PervasiveSensorManager(getContext());
43          mSensorManager.addSensor(Sensor.TYPE_ACCELEROMETER);
44          mSensorManager.addSensor(Sensor.TYPE_LIGHT);
45          mSensorManager.addSensor(Sensor.TYPE_ORIENTATION);
46          mSensorManager.addSensor(Sensor.TYPE_PROXIMITY);
47          mSensorManager.setSensorsCallback(createSensorCallbacks());
48      }
49
50      /**
51       * Register a sensor based on current page.
52       * @param page Indicates the tab that is being showed to the user.
53       */
54      private void registerSensorListener(final int page) {
55          mPage = page;
56          switch (page) {
57              case 0:
58                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
59                  break;
60              case 1:
61                  mCurDrawable = R.drawable.light_sensor_icon;
62                  break;
63              case 2:
64                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
65                  break;
66              case 3:
67                  mCurDrawable = R.drawable.proximity_sensor_icon;
68                  break;
69          }
70      }
71
72      // Inflate the view for the fragment based on layout XML
```

```
73      @Override
74      public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
75                               final Bundle savedInstanceState) {
76          final View view = inflater.inflate(R.layout.pager_item, container
                , false);
77          ivIcon = (ImageView) view.findViewById(R.id.sensor_img);
78          ivIcon.setImageResource(mCurDrawable);
79          final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
80          sensorValues = (TextView) view.findViewById(R.id.
                item_sensor_values);
81          tvTitle.setText(mTitle);
82          return view;
83      }
84
85      @Override
86      public void onDestroyView() {
87          super.onDestroyView();
88      }
89
90      private SensorCallback createSensorCallbacks(){
91          return new SensorCallbackAdapter() {
92
93              @Override
94              public void onReceiveAccelerometerData(float x, float y,
                    float z) {
95                  if(mPage == 0){
96                      sensorValues.setText("x: " + x +"\n y: " + y +"\n z:
                            "+ z);
97                  }
98              }
99
100             @Override
101             public void onReceiveLightData(float light) {
102                 if(mPage == 1){
103                     sensorValues.setText("(x < " + light + ")");
```

```
104                     }
105                 }
106
107                 @Override
108                 public void onReceiveOrientationData(float orientation) {
109                     if(mPage == 2){
110                         ivIcon.setRotation(orientation);
111                     }
112                 }
113
114                 @Override
115                 public void onReceiveProximityData(boolean isNear) {
116                     if (mPage == 3) {
117                         if (isNear) {
118                             ivIcon.setImageResource(R.drawable.
                                    proximity_sensor_icon);
119                         } else {
120                             ivIcon.setImageResource(R.drawable.
                                    proximity_sensor_icon_far);
121                         }
122
123                     }
124                 }
125             };
126     }
127 }
```

Código Fonte D.4: Código criado pelo desenvolvedor 4.

```
1  package com.example.android.common.fragments;
2
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.View;
8  import android.view.ViewGroup;
9  import android.widget.ImageView;
```

```
10 import android.widget.TextView;
11 import com.example.android.slidingtabsbasic.R;
12 import development.master.com.pervasivesensorlib.PervasiveSensorManager;
13 import development.master.com.pervasivesensorlib.SensorCallback;
14 import development.master.com.pervasivesensorlib.SensorCallbackAdapter;
15
16 public class SensorFragment extends android.support.v4.app.Fragment {
17
18     private String mTitle;
19     private TextView mSensorValues;
20     private int mCurDrawable;
21     private int curPage = 0;
22     private PervasiveSensorManager psManager;
23
24     // newInstance constructor for creating fragment with arguments
25     public static SensorFragment newInstance(final int page, final String
           title) {
26         final SensorFragment fragment = new SensorFragment();
27         Bundle args = new Bundle();
28         args.putInt("someInt", page);
29         args.putString("someTitle", title);
30         fragment.setArguments(args);
31         return fragment;
32     }
33
34     // Store instance variables based on arguments passed
35     @Override
36     public void onCreate(final Bundle savedInstanceState) {
37         super.onCreate(savedInstanceState);
38         curPage = getArguments().getInt("someInt", 0);
39         mTitle = getArguments().getString("someTitle");
40         registerSensorListener(curPage);
41         psManager = new PervasiveSensorManager(getContext());
42         psManager.addSensor(Sensor.TYPE_ACCELEROMETER);
43         psManager.addSensor(Sensor.TYPE_LIGHT);
44         psManager.addSensor(Sensor.TYPE_ORIENTATION);
45         psManager.addSensor(Sensor.TYPE_PROXIMITY);
```

```
46          psManager.setSensorsCallback(sensorCallback);
47      }
48
49      /**
50       * Register a sensor based on current page.
51       * @param page Indicates the tab that is being showed to the user.
52       */
53      private void registerSensorListener(final int page) {
54          switch (page) {
55              case 0:
56                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
57                  break;
58              case 1:
59                  mCurDrawable = R.drawable.light_sensor_icon;
60                  break;
61              case 2:
62                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
63                  break;
64              case 3:
65                  mCurDrawable = R.drawable.proximity_sensor_icon;
66                  break;
67          }
68      }
69
70      // Inflate the view for the fragment based on layout XML
71      @Override
72      public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
73                               final Bundle savedInstanceState) {
74          final View view = inflater.inflate(R.layout.pager_item, container
                , false);
75          final ImageView ivIcon = (ImageView) view.findViewById(R.id.
                sensor_img);
76          ivIcon.setImageResource(mCurDrawable);
77          final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
78          tvTitle.setText(mTitle);
```

```
79          mSensorValues = (TextView) view.findViewById(R.id.
                item_sensor_values);
80          return view;
81      }
82
83      @Override
84      public void onDestroyView() {
85          super.onDestroyView();
86      }
87
88      private SensorCallback sensorCallback = new SensorCallbackAdapter() {
89          @Override
90          public void onReceiveAccelerometerData(float x, float y, float z)
                {
91              super.onReceiveAccelerometerData(x, y, z);
92              if(curPage == 0)
93                  mSensorValues.setText("X: " + x + "\nY: " + y + "\nZ: " +
                        z);
94          }
95
96          @Override
97          public void onReceiveLightData(float light) {
98              super.onReceiveLightData(light);
99              if(curPage == 1)
100                 mSensorValues.setText("L: " + light);
101         }
102
103         @Override
104         public void onReceiveOrientationData(float orientation) {
105             super.onReceiveOrientationData(orientation);
106             if(curPage == 2)
107                 mSensorValues.setText("O: " + orientation);
108         }
109
110         @Override
111         public void onReceiveProximityData(boolean isNear) {
112             super.onReceiveProximityData(isNear);
```

```
113             if(curPage == 3)
114                 mSensorValues.setText("P: " + isNear);
115         }
116     };
117 }
```

Código Fonte D.5: Código criado pelo desenvolvedor 5.

```
1  package com.example.android.common.fragments;
2
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.View;
8  import android.view.ViewGroup;
9  import android.widget.ImageView;
10 import android.widget.TextView;
11 import com.example.android.slidingtabsbasic.R;
12 import development.master.com.pervasivesensorlib.PervasiveSensorManager;
13 import development.master.com.pervasivesensorlib.SensorCallback;
14 import development.master.com.pervasivesensorlib.SensorCallbackAdapter;
15
16 public class SensorFragment extends android.support.v4.app.Fragment {
17
18     private String mTitle;
19     private int mCurDrawable;
20     private TextView mitem_sensor_values;
21     private PervasiveSensorManager mSensorManager;
22     private int mpage;
23
24     // newInstance constructor for creating fragment with arguments
25     public static SensorFragment newInstance(final int page, final String
           title) {
26         final SensorFragment fragment = new SensorFragment();
27         Bundle args = new Bundle();
28         args.putInt("someInt", page);
29         args.putString("someTitle", title);
```

```
30          fragment.setArguments(args);
31          return fragment;
32      }
33
34      // Store instance variables based on arguments passed
35      @Override
36      public void onCreate(final Bundle savedInstanceState) {
37          super.onCreate(savedInstanceState);
38          final int curPage = getArguments().getInt("someInt", 0);
39          mTitle = getArguments().getString("someTitle");
40
41          mSensorManager = new PervasiveSensorManager(getContext());
42          mSensorManager.addSensor(Sensor.TYPE_ACCELEROMETER);
43          mSensorManager.addSensor(Sensor.TYPE_LIGHT);
44          mSensorManager.addSensor(Sensor.TYPE_ORIENTATION);
45          mSensorManager.addSensor(Sensor.TYPE_PROXIMITY);
46          mSensorManager.setSensorsCallback(createSensorsCallbacks());
47          registerSensorListener(curPage);
48      }
49
50      /**
51       * Register a sensor based on current page.
52       * @param page Indicates the tab that is being showed to the user.
53       */
54      private void registerSensorListener(final int page) {
55          mpage=page;
56          switch (page) {
57              case 0:
58                  mCurDrawable = R.drawable.accelerometer_sensor_icon;
59                  break;
60              case 1:
61                  mCurDrawable = R.drawable.light_sensor_icon;
62                 break;
63              case 2:
64                  mCurDrawable = R.drawable.orientation_sensor_field_icon;
65                  break;
66              case 3:
```

```
67                  mCurDrawable = R.drawable.proximity_sensor_icon;
68                  break;
69          }
70      }
71
72      private SensorCallback createSensorsCallbacks(){
73          return new SensorCallbackAdapter() {
74              @Override
75              public void onReceiveAccelerometerData(float x, float y,
                    float z) {
76                  super.onReceiveAccelerometerData(x, y, z);
77                  if(mpage==0) {
78                      mitem_sensor_values.setText("#X: " + x + " #Y: " + y
                            + " #Z: " + z);
79                  }
80              }
81
82              @Override
83              public void onReceiveLightData(float light) {
84                  super.onReceiveLightData(light);
85                  if(mpage==1) {
86                      mitem_sensor_values.setText("Light:? " + light);
87                  }
88              }
89
90              @Override
91              public void onReceiveOrientationData(float orientation) {
92                  super.onReceiveOrientationData(orientation);
93                  if(mpage==2) {
94                      mitem_sensor_values.setText("Orientation:? " +
                            orientation);
95                  }
96              }
97
98              @Override
99              public void onReceiveProximityData(boolean isNear) {
100                 super.onReceiveProximityData(isNear);
```

```
101                 if(mpage==3) {
102                     mitem_sensor_values.setText("IsNear:? " + isNear);
103                 }
104             }
105         };
106     }
107
108     // Inflate the view for the fragment based on layout XML
109     @Override
110     public View onCreateView(final LayoutInflater inflater, final
            ViewGroup container,
111                              final Bundle savedInstanceState) {
112         final View view = inflater.inflate(R.layout.pager_item, container
                , false);
113         final ImageView ivIcon = (ImageView) view.findViewById(R.id.
                sensor_img);
114         ivIcon.setImageResource(mCurDrawable);
115         final TextView tvTitle = (TextView) view.findViewById(R.id.
                item_title);
116         tvTitle.setText(mTitle);
117         mitem_sensor_values = (TextView) view.findViewById(R.id.
                item_sensor_values);
118
119         return view;
120     }
121
122     @Override
123     public void onDestroyView() {
124         super.onDestroyView();
125     }
126 }
```