UNISINOS

Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Guilherme Callegaro Sesterheim

Ambiente móvel para otimização de desktop semântico

São Leopoldo, 2013

S494a Sesterheim, Guilherme Callegaro

Ambiente móvel para otimização de desktop semântico / Guilherme Callegaro Sesterheim. - 2013.

77 f. : il. ; 30cm.

Dissertação (mestrado) -- Universidade do Vale do Rio dos Sinos. Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2013.

Orientador: Prof. Dr. Sérgio Crespo.

1. Computação. 2. Dispositivo móvel - Interface - Desktop. 3. Software de aplicação - Desenvolvimento. I. Título. II. Crespo, Sérgio.

CDU 004

Catalogação na Publicação:

Bibliotecário Eliete Mari Doncato Brasil - CRB 10/1184

UNIVERSIDADE DO VALE DO RIO DOS SINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO

EM COMPUTAÇÃO APLICADA

NÍVEL MESTRADO

Guilherme Callegaro Sesterheim

Ambiente móvel para otimização de desktop semântico

São Leopoldo

2013

Guilherme Callegaro Sesterheim

Ambiente móvel para otimização de desktop semântico

Dissertação de mestrado apresentada para a obtenção do título de Mestre pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos

Orientador:

Prof. Dr. Sérgio Crespo

São Leopoldo

Fevereiro de 2013

RESUMO

Este trabalho foi desenvolvido com o objetivo de melhorar a experiência do usuário com o desktop de dispositivos que usam sistema operacional Android através de identificação de contextos de uso. Os contextos de uso foram identificados através de registros de uso de aplicativos durante utilização dos mesmos. Para tanto, diferentes técnicas foram estudadas, tanto para entender a localização do dispositivo, como também para identificar e analisar diversos trabalhos relacionados de diferentes áreas.

Para entender a localização do dispositivo, diferentes hardwares e as maneiras de interagir com os mesmos foram estudadas. Hardwares de Bluetooth, Wireless, GPS, NFC, Infravermelho e acelerômetro foram estudados e sua interação foi testada para obter informações e então armazená-las. Trabalhos da área também foram estudados, tendo desde dispositivos criados com o propósito de identificar localização, até alterações no comportamento de aplicações que estamos acostumados a usar com o propósito de deixa-las mais específicas.

Como resultado deste trabalho, um protótipo funcional foi criado com o propósito de melhorar a experiência do usuário com o desktop de seu smartphone. Uma maneira inteligente de arranjar e destacar os ícones do desktop, e também de alertar o usuário sobre recomendações de aplicativos baseadas em histórico de uso, foi feita para ajuda-lo com suas tarefas usuais.

ABSTRACT

This work was developed aiming to improve the user experience in mobile devices' desktop that uses Android operational system using context identification. The use contexts were identified through application use registers created during their utilization. For that, different techniques were used, such as to understand the device's localization, and also to identify the related works in different problems found in different areas.

To understand the device's localization, different hardwares and the ways to interact with them were studied. Bluetooth, Wireless, GPS, NFC, Infrared and accelerometer hardware were searched and their interaction were tested to obtain and to store their information. Works related to the area were studied, finding since entire devices created specifically to identify the localization, and other ones created to change the application's behavior we are used to use with the purpose of making it more specific.

As result of this work, a functional prototype was created to enhance the user experience with his smartphone's desktop. A more intelligent way to rearrange and to detach the desktop icons, and also to alert the user about the recommendation, was made to help him with his usual tasks.

LISTA DE TABELAS

LISTA DE IMAGENS

SUMÁRIO

1. Introdução

Como descrito por (Sang-Hak & Tae-Choong, 2004), o campo de aplicações móveis sensíveis a contextos possui um grande potencial de explorar cenários que facilitariam tarefas simples e corriqueiras de usuários comuns. O campo de localização de dispositivos móveis é explorado por várias áreas da tecnologia (como descrito por (Want et al., 1995) e (Chu, Kansal, Liu, & Zhao, 2012)), e conhecer a localização de um dispositivo é útil e necessário para muitas aplicações que podem interagir com nosso dia-a-dia, tais como: sistemas de recomendações de atividades próximas a locais em que um usuário estiver, sistemas de mapas em menor escala do que os já conhecidos com GPS, marketing direcionado ao local e horário em que o indivíduo se encontra, etc ((Sang-Hak & Tae-Choong, 2004) e (Gui et al., 2009)).

O termo *context-aware* (sensibilidade a contextos), que se refere a permitir que o ambiente possa interferir no comportamento de uma aplicação, apareceu pela primeira vez em (Theimer, 1994). Contexto é definido por (Yamin, 2004) como toda informação necessária à aplicação que pode ser obtida a partir do hardware computacional, onde qualquer alteração em seu estado dispara um processo de adaptação na aplicação. Utilizando esta visão, o contexto permite enfocar e utilizar os aspectos relevantes para uma situação particular e, caso seja necessário, ignorar outros. Uma aplicação sensível a contextos tem explicitamente a tarefa de identificar e definir as características de uma situação e essas passam a integrar o seu contexto (Lopes, 2006).

Como parte do trabalho de uma aplicação, conhecer a localização do dispositivo é importante para guardar o histórico de uso gerado pelo usuário. Para exemplificar a complexidade de se identificar a localização de um dispositivo, é destacada a diferença entre as alternativas de se obter a localização do usuário para ambientes fechados e ambientes abertos: a) para ambientes abertos a tecnologia de GPS se mostra usual e suficiente, pois é utilizada largamente para tal finalidade e provê bons e confiáveis dados. b) já para ambientes fechados, a localização dos usuários torna-se menos trivial devido ao raio de erro que as tecnologias costumam apresentar. Para este problema, várias tecnologias bastante conhecidas (*hardwares*), como GPS, Wi-Fi (IEEE 802.11), Bluetooth, NFC, Infravermelho e acelerômetro estão apresentadas e foram utilizadas. Junto destes componentes descritos, um chip menos conhecido, o LPS331AP (que se trata de um barômetro bastante preciso), mas com grande potencial de uso também será apresentado para facilitar a localização do usuário. Este chip está presente em aparelhos difundidos do mercado, mas ainda não é largamente utilizado.

As tecnologias para localização usadas neste trabalho (seção 2.5) foram escolhidas com base no conhecimento já possuído pelo autor sobre as mesmas através de trabalhos anteriores como (Sesterheim, 2011) e sobre a fácil implementação relacionada a elas na plataforma Android, demonstrada através de alguns experimentos na seção 2.7. Este trabalho apresenta trabalhos relacionados à área de localização de dispositivos móveis (seção 3), de maneira a garantir conhecimento sobre os adaptadores (*hardwares*) mais difundidos no mercado e que foram utilizados durante o desenvolvimento da dissertação. Além

disto, apresenta trabalhos que possuem ideias semelhantes e que possuem o conceito de sensibilidade a contextos (seção 3.1).

1.1 Contextualização do problema

Segundo a Forbes por (Lopez, 2012), em 2015 teremos 959 milhões de dispositivos móveis (804 milhões de smartphones e 149 milhões de tablets). Por esta crescente demanda de mobilidade e a tendência da busca por atualizações em qualquer lugar vindas de todos os usuários, diferentes sistemas de interação com os mesmos são idealizados para diferentes finalidades, tanto comerciais quanto não comerciais.

Em recente trabalho semelhante pela Microsoft, de (Chu et al., 2012), um diferente comportamento no sistema operacional, semelhante a esta proposta foi apresentado – este foi apresentado por ter sido a proposta encontrada que mais possui condições de se disseminar rapidamente. O trabalho (com detalhes na seção 3.3), diferentemente deste idealizado, propõe que o sistema operacional se comporte de maneira a prever utilizações do usuário e previamente carregar aplicativos em segundo plano, tornando a abertura do mesmo mais rápida. Esta aproximação de uma reconhecida empresa do mercado para esta área ajuda a reforçar a importância da otimização de dispositivos móveis e de seus aplicativos de acordo com os gostos do usuário.

O problema encontrado foi a possibilidade de melhorar o comportamento do desktop de dispositivos móveis através de recomendações criadas com o uso de dados de histórico de uso do próprio usuário. Por isto, esta dissertação possui como objetivo apresentar os meios de pesquisa e fundamentações utilizadas para desenvolver um ambiente de otimização de execução de aplicativos para sistemas móveis através de desktop semântico.

O foco deste trabalho é a ponta de interação com o usuário (a interface), que abrange a organização do desktop e rearranjo em seus ícones, melhorias com o consumo da bateria e a geração de dados históricos sobre o uso dos aplicativos instalados.. As tarefas de coletar dados de localização, recomendações e armazenamento de dados de uso estão cobertas pelos outros projetos deste programa (descritos no item 5.1).

1.2 Contextualização do projeto

Este projeto foi desenvolvido em paralelo a outros dois projetos. Este programa está sob coordenação do professor Dr. Sérgio Crespo e está sendo desenvolvido junto ao programa de mestrados do PIPCA (Programa Interdisciplinar de Pós-graduação em Computação Aplicada - Unisinos). Os três projetos, denominados CONTEXT SERVER, CONTEXT MANAGER, e INTERFACE SENSÍVEL AO CONTEXTO, sendo que este último é tratado por este trabalho, possuem as seguintes funcionalidades descritas abaixo:

1.2.1 Projetos CONTEXT MANAGER e CONTEXT SERVER

Com esta seção, objetiva-se descrever resumidamente quais recursos auxiliares a este projeto serão abordados pelos outros trabalhos do programa.

O projeto CONTEXT MANAGER pode ser descrito, rapidamente, como:

- Responsável por utilizar os sensores existentes em cada aparelho (descritos pela seção de hardwares estudados – 2.5), identificar as ações que o usuário toma e guardar estes dados, através do Context Server, como histórico para fazer recomendações.
- A partir do histórico criado (e armazenado no CONTEXT SERVER), recomendar alterações no desktop de cada aparelho.

Já o projeto CONTEXT SERVER pode ser descrito, rapidamente, como:

- Responsável por receber os dados de uso do usuário como: sensores, aplicações executadas, horário de atividade, etc. e guardá-los para poder prover dados ao CONTEXT MANAGER para então gerar recomendações.

1.2 Objetivo da Pesquisa

Através do desenvolvimento deste trabalho, objetivou-se responder à seguinte questão: **como se pode tornar mais rápido o uso de um smartphone através da interferências em seu desktop?**

O objetivo principal desta proposta é o desenvolvimento de uma arquitetura para otimização de uso do desktop de smartphones que usem o sistema Android, que será ativado a partir de recomendações vindas de um agente de software terceiro, e também identificando e registrando usos de quaisquer aplicativos que o usuário faça. Como efeito natural, uma otimização sobre o consumo de bateria foi notada e mostrada também como aspecto deste projeto (seção 2.2).

A escolha do sistema Android foi feita através de uma comparação entre os sistemas líderes de mercado, que está detalhada na seção 2.4, e que também mostra quais sistemas operacionais são capazes de receber a proposta.

Tendo este objetivo geral em mente, os seguintes objetivos específicos foram definidos:

- Projetar uma arquitetura flexível para remanejamento de informações no desktop, que não seja vinculada a um sistema operacional, e que sua ideia de implementação possa ser usada por quaisquer outros SOs.
- Compreender o funcionamento de diferentes aplicativos de desktop usados nos dispositivos móveis.
- Desenvolver o protótipo usando agentes de software para comunicação externa.
- Testar e validar o uso do aplicativo.
- Projetar uma arquitetura para adaptação da interface do desktop em relação ao seu contexto.

1.3 Organização do texto

O texto desta proposta está apresentado nas seguintes sessões:

2) Fundamentação teórica – apresenta os estudos feitos sobre os assuntos envolvidos. Estudos que serão utilizados como embasamento para o desenvolvimento e limitação do escopo do projeto.

3) Trabalhos relacionados – serão apresentados trabalhos da área que possuem foco parecido com o desta proposta e/ou que usam os mesmos conceitos que esta proposta abordará.

4) Notações – capítulo responsável por explicar e facilitar o entendimento sobre os diagramas incluídos nesta proposta, que estão apresentados em uma notação reconhecidamente simplificada.

5) Proposta – a proposta de desenvolvimento da dissertação.

6) Desenvolvimento – capítulo que apresenta o ambiente e como foi desenvolvido.

7) Validação – capítulo que apresenta a validação do desenvolvimento feito.

8) Cronograma – cronograma apresentando como o desenvolvimento do ambiente foi gerenciado.

9) Trabalhos futuros – seção indicando projetos futuros relacionados ao que foi gerado neste trabalho.

10) Bibliografia – material estudado para desenvolver este projeto.

2. Fundamentação tecnológica

Nesta seção estão apresentados os estudos das tecnologias, tanto de software quanto hardware, que foram utilizadas no projeto. As tecnologias usadas no projeto são tanto de embasamento teórico (estudo do hardware utilizado), quanto com fundamentos técnicos (considerações sobre a linguagem e suas formas de implementação).

2.1 Conceitos

Esta seção apresenta importantes conceitos encontrados durante a pesquisa bibliográfica, que foram mantidos em uso durante o desenvolvimento e pesquisa de materiais para o ambiente. Estes conceitos foram apresentados para dar clareza a como a comunicação entre os projetos (apresentados na seção 5) acontece e também sobre quais pontos precisaram ser abordados pela pesquisa. As seções 2.1.2 e 2.1.3 apresentam conceitos relativos à localização que foram usados para a criação de um registro de uso, mostrado na seção 6.3.2.

2.1.1 **Contexto:** como base para a pesquisa, o importante conceito de contexto foi definido: na área pesquisada, é definido como um conjunto de informações que são utilizadas para representar logicamente situações do mundo real. Pode também ser descrito como um espaço multidimensional caracterizado por diferentes listas de valores relevantes gerados por cada informação "sentida" (sensores, entrada de dados do usuário, etc) (Padovitz, Loke, & Zaslavsky, 2008).

2.1.2 **Uma localização não precisa, necessariamente, possuir dados geográficos como latitude e longitude.** Uma localização pode ser determinada pelo simples rótulo (*label*) de "escritório", "casa", ou "escola", por exemplo. Para o primeiro exemplo, o contexto "escritório" de um usuário qualquer poderia ser identificado, por exemplo, por um determinado número e nome de conexões wireless disponíveis, somado à força de sinal de cada uma delas, ou ainda por certo número de dispositivos *bluetooth* ligados previamente detectados (Johnson, 2007).

2.1.3 **A impressão digital** por (Johnson, 2007) foi um conceito muito relevante encontrado em trabalhos da área. Este conceito propõe o mapeamento de uma ou mais características altamente específicas que podem ajudar a determinar uma localização. Como exemplo temos novamente o contexto "escritório", que poderia ser determinado pela existência das redes wireless "nomeCompanhia1" e "nomeCompanhia2" e pela NÃO detecção do *bluetooth* do celular da esposa do empregado em questão (que seria identificado baseado em histórico de uso). Este tipo de conceito poderia ser aplicado a todos os contextos conhecidos, diferenciando apenas os dados que seriam utilizados para determinar a impressão digital.

As impressões digitais poderiam ser criadas de duas diferentes maneiras: a) manualmente pelo usuário: onde o mesmo indicaria em uma interface quais são as

características (sensores, dados trafegados, etc) existentes no contexto em que ele está sendo inserido juntamente de seus status, e desta maneira acabaria criando a impressão digital. b) detecção automática através da inteligência da aplicação: através dos logs de histórico dos usuários, a própria aplicação poderia criar as impressões digitais e passar a usá-las detectando e apenas questionando o usuário se aquela é uma "impressão digital" válida ou não.

Utilizando a segunda maneira descrita, as impressões poderiam ser alteradas pelo usuário caso o mesmo não concorde com as características de criação de alguma delas, ou ainda discorde de sua especificidade ou falta dela. Ainda sobre esta mesma impressão, criada por um usuário qualquer, poderia ser usada, após alguma adaptação, por outros diferentes usuários que estejam inseridos no mesmo contexto. Desta maneira, os dispositivos destes novos usuários encontrariam as impressões digitais já existentes nos aparelhos de seus colegas de trabalho, por exemplo. Estas "impressões digitais copiadas" deveriam ter o cuidado de apropriar-se de características genéricas do ambiente tais como (seguindo a lógica do exemplo do escritório) as redes wireless, e descartariam a detecção do bluetooth da esposa do usuário com a impressão digital original, por exemplo.

2.2 Bateria

Um problema comum encontrado no tipo de framework estudado e na grande maioria das bibliografias consultadas, como (Weerasinghe & Warren, 2010) e (Kostakos & O'Neill, 2007), é o consumo de bateria por parte da aplicação. Diferentemente dos hardwares de *bluetooth*, acelerômetro, RFID, e outros, os de GPS e Wireless consomem uma grande quantidade de energia, e com isto podem tornar o uso do aplicativo algo oneroso ao usuário ou ainda que lhe traria algum transtorno, o que criaria uma má impressão sobre o funcionamento do mesmo.

Uma proposta identificada e capaz de minimizar este impacto (Weerasinghe & Warren, 2010), (Gashti, Pujolle, & Rotrou, 2009) seria tornando a utilização dos recursos menos consumistas mais frequente, e em contrapartida utilizar os recursos que mais consomem energia com uma periodicidade maior e em um determinado espaço de tempo. Também enquanto os sensores não estiverem sendo usados, eles devem ser mantidos desligados para economizar ainda mais bateria.

Mesmo não sendo o foco deste trabalho, o consumo de bateria foi uma preocupação e se verificou um ganho em seu uso, através da melhoria natural das ações tomadas pelo usuário destacado na seção 7.4.

Sabe-se que o consumo de bateria de dispositivos móveis, é muito expressivo enquanto o usuário está com a tela do mesmo ligada. Segundo (Korhonen, 2011), a figura 1 abaixo apresenta os principais consumidores de bateria em dispositivos móveis.

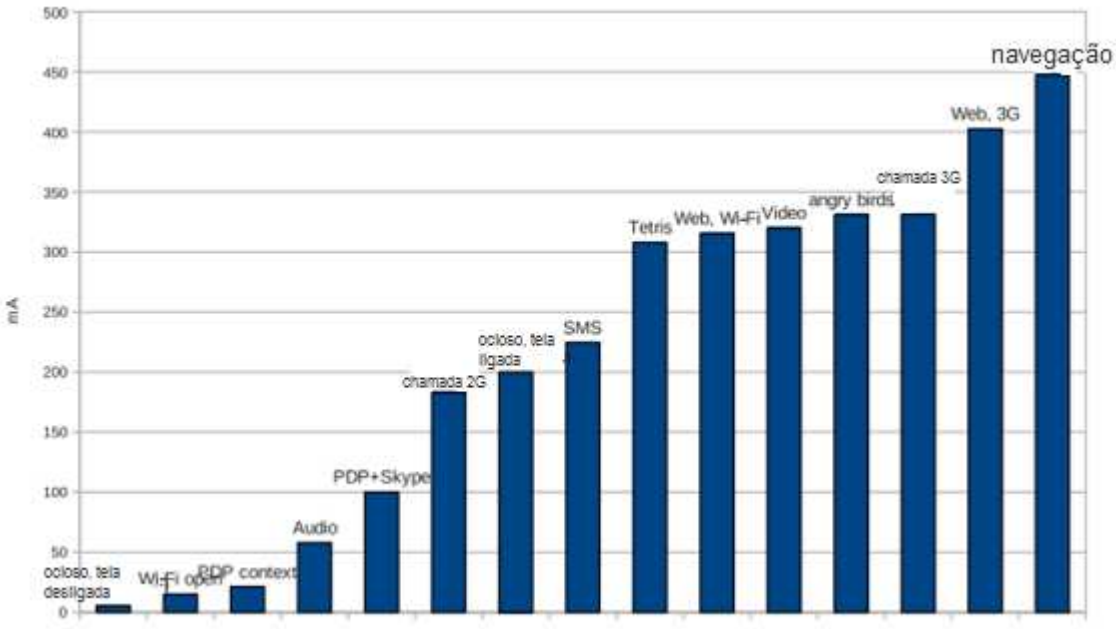


Figura 1 - Consumo de bateria por atividade. Fonte (Korhonen, 2011)**.**

Com esta figura, pode-se verificar que a navegação dentro do aplicativo é responsável pelo maior fator consumista de bateria.

Tendo este fator em vista, este trabalho apresentará uma redução no consumo de bateria de dispositivos móveis, além de seus focos principais. O detalhamento sobre o ganho no consumo de bateria esperado pode ser visto no item 7.4.

2.3 Agentes de software

Agentes de software são sistemas de computador capazes de tomar ações autônomas ou não. Sua interação com o ambiente pode ser descrita essencialmente como: o agente recebe uma entrada (uma chamada), processa os dados enviados, e retorna uma saída (resposta) para o ambiente. Agentes podem possuir características de autonomia, pró-atividade e sociabilidade (Silva & Castro, 2005).

Agentes foram utilizados neste projeto devido a sua simples utilização como interface para outros sistemas. Como mostrado na figura 8, seção 5.3.1, os agentes foram usados neste projeto para criar comunicação entre os diferentes sistemas que o compõem.

Na figura 2, abaixo, temos o exemplo de utilização de um agente, onde: o usuário fará uma requisição ao mesmo, que servirá de mensageiro com a aplicação. Após receber a requisição, ela será processada, e a resposta encontrada devolvida ao usuário.

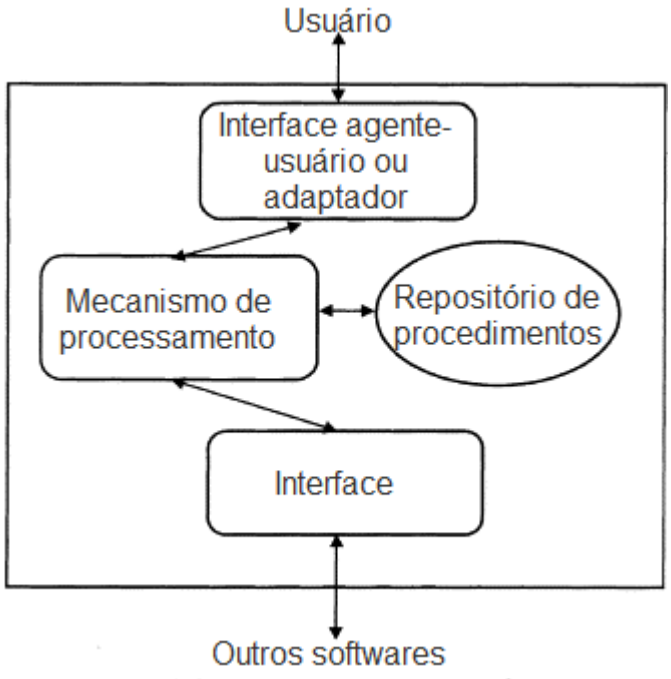


Figura 2 - Exemplo de utilização de agente. Fonte: (Norvig & Stuart J, 1988)

A seguir estão descritos quatro tipos de agentes que interagem de maneiras diferentes com o ambiente que os executa (Norvig & Stuart J, 1988).

- **Agentes reativos simples:** são tipos comuns de agentes, que executam ações ignorando quaisquer históricos.
- **Agentes reativos baseados em modelos:** o agente mantém histórico sobre suas execuções, o que se reflete em seus novos estados.
- **Agentes baseados em objetivos:** aliado dos dados de histórico, o agente também possui informações sobre quais são seus objetivos de execução, e os utiliza nas mesmas.
- **Agentes baseados em utilidade:** junto dos dados de histórico e dos objetivos, o agente mapeia uma sequência de resultados de execução, que cria um índice de satisfação de execuções. Este índice é utilizado em futuras execuções para tentar assemelhar-se ao histórico de índices possuído.

2.4 Sistema operacional

Os principais sistemas operacionais móveis, líderes de mercado (conforme figura 3 abaixo), foram estudados para ajudar na definição deste projeto. O estudo foi executado para encontrar embasamento para escolher o sistema que mais facilmente difundirá o ambiente criado e por isto recebeu a implementação. Com este tudo, quatro alternativas foram encontradas:

- Android
- iOS

- BlackBerry OS
- Windows Phone

A seleção entre eles, que resultou na escolha do Android, foi feita com base em 3 aspectos fundamentalmente escolhidos tendo em vista a qualidade e futura facilidade de difusão do projeto, que podem ser vistos na tabela 1 abaixo:

	Android	**iOS**	**BlackBerry OS**	**Windows Phone**
Market Share	Maior Market share entre os apresentados (figura 2)	-	-	-
Velocidade de atualização do SO	Velocidade de atualização da plataforma mais rápida entre os apresentados (XCubeLabs, 2012)	-	-	-
Viabilidade de implantação do projeto idealizado	Possível	Impossível – o kit de desenvolvimento do iOS não prevê alteração no desktop	? – kit de desenvolvimento indisponível no momento da criação deste trabalho	Impossível – o kit de desenvolvimento do Windows phone não prevê alteração no desktop

Tabela 1 – Tabela comparativa entre sistemas operacionais possíveis para o projeto

Para os sistemas iOS e Windows Phone, a viabilidade de implementação deste projeto, como foi proposto, ficou definida como inviável devido a restrições de suas mantenedoras. Para ambos os sistemas, as responsáveis não permitem que o desktop seja modificado. Para o Blackberry OS a informação não foi encontrada, pois no momento de escrita deste trabalho a RIM (Research In Motion) passa por uma reestruturação de seu SO e ainda não divulgou dados detalhados sobre o desenvolvimento de aplicativos deste tipo.

O trabalho proposto, ainda com as restrições impostas por Microsoft (responsável pelo Windows Phone) e Apple (responsável pelo iOS), poderá ser desenvolvido para seus respectivos sistemas, desde que algumas alterações sejam feitas em sua arquitetura.

2.4.1 Android

Android é um sistema operacional móvel que é executado sobre Linux, desenvolvido e mantido pela Google desde 2005 e lançado em 2008. Permite aos desenvolvedores criarem aplicações na linguagem Java, C, C++ e C#, e atualmente é o sistema operacional móvel líder de ativações e de utilização no mercado (Tecnologia – acesso em 12/11/2011).

O Android foi selecionado para desenvolver o ambiente idealizado neste projeto tendo em vista algumas vantagens que estão descritas abaixo:

- Atualmente o Android, como pode ser visto com a figura 3, é o sistema operacional móvel que é líder de mercado, e por este motivo proporcionará ao projeto uma maior visibilidade devido a grande base de usuários de que ele dispõe.

Venda mundial de dispositivos para usuários finais por plataforma no último trimestre de 2012 (milhares de unidades)

Sistema operacional	3Q12 Unidades	3Q12 Market share (%)	3Q11 Unidades	3Q11 Market Share (%)
Android	122,480.0	72.4	60,490.4	52.5
iOS	23,550.3	13.9	17,295.3	15.0
Research In Motion	8,946.8	5.3	12,701.1	11.0
Bada	5,054.7	3.0	2,478.5	2.2
Symbian	4,404.9	2.6	19,500.1	16.9
Microsoft	4,058.2	2.4	1,701.9	1.5
Outros	683.7	0.4	1,018.1	0.9
Total	**169,178.6**	**100.0**	**115,185.4**	**100.0**

Source: Gartner (November 2012)

Figura 3 - Market share do Android. Comparação entre o terceiro trimestre de 2012 e 2011

- O sistema operacional é atualizado constantemente (em média a cada 5 meses, segundo (XCubeLabs, 2012)) pela mantenedora, e estas atualizações não trazem problemas para aplicações antigas, de maneira que não é necessário adequar desenvolvimentos já finalizados
- O sistema operacional, a partir da versão 3.0, também é utilizado por *tablets* (XCubeLabs, 2012), que também garantirão uma maior difusão da ferramenta, visto que poderá ser utilizada em mais plataformas devido a sua compatibilidade com o SO.
- Possui rica documentação disponível na web, que possibilita um desenvolvimento ágil, e também um fácil entendimento a futuros interessados em dar manutenção na aplicação que foi desenvolvida.
- As aplicações podem ser desenvolvidas a partir de vários sistemas operacionais, tais como Windows, Linux, OS X, etc., com poucas restrições deste gênero.

2.5 TAM (Technical Arquitecture Modeling)

TAM é um padrão de modelagem desenvolvido pela SAP, baseado em MOF (Meta-Object Facility) 2.0 e na UML (Unified Modeling Language), com o objetivo de simplificar representações de arquiteturas de sistemas. (SAP, 2007).

O TAM foi utilizado para desenhar os fluxos de informações das aplicações do projeto e também para criar suas imagens de arquiteturas, que estão apresentadas a seguir na seção cinco.

A tabela 2 foi criada para mostrar a finalidade de cada um dos objetos do TAM que foram utilizados neste projeto:

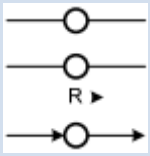
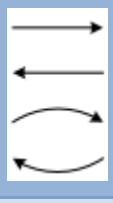
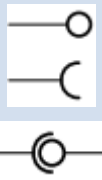
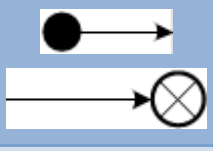
Elemento	Descrição
Agentes 	Elementos ativos que são capazes de desempenhar tarefas. Agentes contêm agentes, armazenamentos, subsistemas, componentes e classes (herdadas).
Canais 	Elementos passivos que são utilizados para comunicação entre agentes. Toda informação transferida é volátil. Opcionalmente, podem ser utilizados para requisição (representados pelo "R ➔") e/ou determinação da direção de fluxo.
Acessos (ler, escrever, alterar) 	Arcos de acesso definem a maneira como agentes podem acessar um elemento.
Interfaces 	Interfaces definem pontos de conexão entre os componentes e seu ambiente. Componentes fornecem um conjunto de serviços (interface fornecedora) e **os conectam** com serviços prestados por terceiros elementos (interface receptora)
Marcadores 	A primeira figura determina onde inicia o fluxo de dados. Já a segunda informa quando o fluxo acaba
Anotações 	Anotações são utilizadas para enriquecer a informação que o fluxo está transmitindo. Caso o autor deseje passar alguma informação que tenha ficado implícita, ou mesmo que tenha ficado de difícil entendimento, uma nota pode ser utilizada.

Tabela 2 – Objetos utilizados do TAM

2.6 Hardware

Nesta seção (e complementarmente na seção 2.6) serão apresentados os hardwares de NFC, Bluetooth, IEE 802.11, GPS, Infravermelho e Acelerômetro, que foram escolhidos para serem estudados. Estes hardwares foram escolhidos por proporcionarem acesso a informações sobre o comportamento e a localização dos usuários, e suas descrições estão junto do relacionamento de eventuais pontos fortes e fracos. Este estudo foi feito para determinar quais hardwares seriam usados durante a implementação e também para servir como base para projetos futuros.

Alguns dos hardwares apresentados nesta seção foram alvos de pequenos experimentos (apresentados na seção 2.7) para identificar possíveis dificuldades em sua utilização no projeto (detalhado na seção 2.6). Os mesmos foram estudados e utilizados para agrupar os dados do contexto do usuário em questão e então enviá-los ao servidor conforme descrito na seção 6.3.2.

2.6.1 NFC

É a sigla para *Near Field Communication* (Comunicação por Proximidade de Campo). O NFC é uma tecnologia de radiofrequência que diferencia-se pela distância de operação, normalmente de 0 a 20 cm entre os dispositivos. A ideia do NFC é tornar a comunicação entre dispositivos mais simples: basta aproximá-los para que isso ocorra, eliminando a necessidade de procedimentos eventualmente complexos como autenticação ou outros. Pelo fato da distância de operação ser curta, a comunicação é inerentemente segura no que diz respeito a tentativas de interceptação, e barrar esse tipo de ação deve ser feito através das medidas usuais de segurança que devem ser implementadas no protocolo de rede e/ou na aplicação (ECMA, 2005). O NFC atua na faixa dos 13.56 MHz, frequência também que é livre de restrições na maioria dos países, com velocidades típicas entre 106 e 424 kbit/s. Em síntese, a tecnologia é utilizada para dois propósitos: a) transmissão de pequena quantidade de dados e b) comunicação secundária entre dispositivos. A especificação NFC prevê ainda que a tecnologia seja compatível com dispositivos RFID (*Radiofrequency Identication* - Identificação por Radiofrequência), que operam na mesma frequência e mesmos modos de operação, especialmente os passivos. Dessa maneira muitos dispositivos habilitados com NFC são capazes de emular cartões com a tecnologia RFID ou mesmo etiquetas (tags) RFID.

2.6.1.1 Problemas do NFC

- Alguns dispositivos, dependendo de sua arquitetura, bloqueiam a operação do hardware do NFC baseado no nível de bateria do aparelho. Isto ocorre pela, em geral, pouca utilização do NFC. Como exemplo, temos os aparelhos Nokia que utilizam Symbian como SO, que barram o NFC quando o nível de bateria cai para menos de 20%, enviando mensagens de erro para as aplicações que o requisitam (Kostakos & O'Neill, 2007).
- Com a difusão do NFC, ocorrendo o uso excessivo e em muitos locais, pode ocorrer algum pareamento e transmissão de dados indesejados com sistemas não previstos. Por

exemplo, um cartão que seria usado para autenticação na portaria de um prédio poderia realizar pareamento por acidente com uma máquina para pagamentos através de NFC que estaria aguardando o celular do usuário, já que celular e carteira (onde o cartão de autenticação poderia ficar) geralmente ficam próximos fisicamente. Isto faz com que este problema precise ser resolvido em nível de software, exigindo interação com o usuário que precisará confirmar o pareamento com determinados sistemas (Kostakos & O'Neill, 2007).

2.6.2 Bluetooth

É uma especificação industrial para a comunicação em curta distância de redes sem fio com um baixo custo e alta operabilidade. A tecnologia de Bluetooth foi projetada inicialmente para suportar redes simples de dispositivos e periféricos pessoais como celulares, PDAs, computadores e mouses. Iniciou em 1998 com um consórcio entre as empresas Ericsson, Nokia, IBM, Intel e Toshiba, chamado Bluetooth SIG (*Special Interest Group*) com o objetivo de expandir e promover o conceito Bluetooth e estabelecer um novo padrão industrial. Assim como o Wi-Fi e outras tecnologias wireless atuais, as preocupações com o Bluetooth incluem a segurança e a interoperabilidade com outros padrões de comunicação (Bluetooth website, 2011).

Os sinais wireless transmitidos em Bluetooth alcançam distâncias curtas, tipicamente de até 10 metros. O protocolo opera na faixa de licença-livre ISM (*Industrial, Scientific, and Medical*) em 2,45 GHz e alcança velocidades de até 723,1 kbit/s. A fim evitar interferência com outros protocolos que podem estar usando a faixa de 2,45 GHz, o protocolo do Bluetooth divide a faixa em 79 canais e muda de canal até 1600 vezes por o segundo.

(Kobayashi, 2004) Os dispositivos Bluetooth têm quatro estados básicos de operação: a) *master* (controla uma rede), b) *active slave* (participante ativo de uma rede), c) *passive slave* (participante de baixa prioridade de uma rede, mas que ainda permanece sincronizado) e d) *standby* (não conectado a rede alguma, e aguardando ocasionais pedidos de outros dispositivos).

2.6.2.1 **Vantagens do Bluetooth** (Kobayashi, 2004)

- É esperado que as tecnologias que utilizam cabo para comunicação próxima tendam a desaparecer com a difusão do Bluetooth, fazendo com que sua utilização seja em um número enorme de dispositivos.
- O consumo de energia do hardware Bluetooth é pequeno, permitindo que dispositivos como smartphones, notebooks e outros saiam de fábrica já com a placa acoplada sem comprometer o tempo de uso do mesmo.
- Pelo baixo consumo de bateria, há a possibilidade de instalar chips transmissores de Bluetooth em quaisquer dispositivos e manter comunicação, como headsets, impressoras, smartphones, HD (harddisk) móveis, etc.

2.6.3 IEEE 802.11

O IEEE (*Institute of Electrical and Eletronics Engineers*) constituiu um grupo de pesquisa para criar padrões abertos de comunicação sem fio. Este projeto nasceu em 1990, mas ficou por aproximadamente sete anos inalterado. A causa principal era a baixa taxa de transferência de dados que a tecnologia inicialmente oferecia (na faixa de kbit/s). Conforme a taxa de transferência de dados passou a atingir a faixa de Mbit/s, a rede sem fio começou a ser vista como uma tecnologia promissora e passou a receber reais investimentos para a construção de equipamentos que possibilitassem a comunicação sem fio entre computadores (IEEE 802.11 website, 2011).

A IEEE 802.11 é um conjunto de padrões para implementação de redes locais sem fio e consiste basicamente de uma série de modulações sobre um mesmo protocolo. As especificações mais utilizadas deste protocolo são a 802.11b e 802.11g, que operam na faixa dos 2.4GHz e foram rapidamente disseminadas. Esta especificação tem como um dos principais objetivos apresentar robustez com relação a bloqueio de interferência. Abaixo temos algumas características do padrão (Nakahati et. Al 2006):

A tabela 3 apresenta a diferença entre cada uma das especificações do 802.11:

Padrão	Taxa de bits	Alcance interno	Freqüência de operação
802.11	Até 2 Mbps	100 metros	2.4 GHz
802.11a	Até 54 Mbps	25 a 100 metros	5 GHz
802.11b	Até 11 Mbps	100 a 150 metros	2.4 GHz
802.11g	Até 54 Mbps	100 a 150 metros	2.4 GHZ

Tabela 3 - Especificações mais difundidas do IEEE 802.11. Fonte: (Nakahati et. Al 2006)

E a figura 4 apresenta o alcance de cada uma delas:

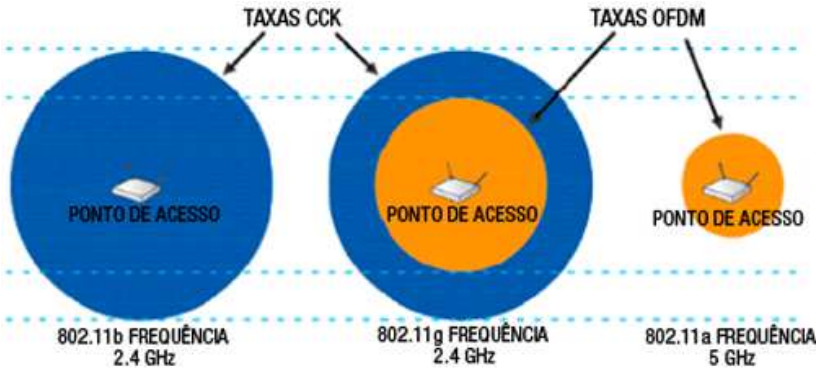


Figura 4 - Alcance dos padrões. Fonte: (Nakahati et. Al 2006)

O IEEE 802.11, hoje em dia, é largamente difundido. É utilizado por muitas pessoas em suas casas e também é muito utilizado em estabelecimentos públicos e privados, como empresas que

pretendem oferecer uma conectividade simplificada a seus empregados e/ou clientes. Mais recentemente a tecnologia também vêm sendo utilizada em locais públicos por governos interessados em prover boa conexão para seus habitantes e, em menor escala, por empresas de telecomunicação interessadas em atrair mais clientes através de ações de marketing.

2.6.4 **GPS** (Alves, 2006)

GPS (Global Positioning System) é um sistema global composto por 24 satélites que orbitam a Terra a uma altura aproximada de 20.000 km, que permitem que dispositivos receptores possam saber sua própria localização com uma boa precisão.

O desenvolvimento do projeto foi iniciado em 1973 pela defesa dos EUA com objetivos militares. Os 24 satélites que orbitam a Terra estão em seis órbitas estáveis e predeterminadas, sendo que a cada 12h completam uma órbita e cada satélite tem 28 graus de visualização sobre a Terra. Este ângulo assegura que qualquer ponto da superfície do planeta esteja sendo visualizada a todo instante por no mínimo 4 satélites, sendo que este número pode chegar a até 10 por alguns momentos.

Cada um dos satélites emite através de ondas de rádio um padrão fixo que é recebido pelos dispositivos na Terra. O receptor então, através de cálculos usando os tempos de recepção e emissão do sinal, verifica qual é a provável localização do dispositivo.

Como aplicações possíveis e já utilizadas para o GPS, algumas se destacam por influenciar diretamente em nossas vidas:

- Meteorologia – o GPS gera informações para a previsão do tempo e estudo do clima.
- Localização para resgates – uso do GPS para guiar helicópteros ou outros meios de transporte até o local onde o socorro é necessário.
- Roteiros de viagens – para determinar a localização dos usuários e identificar pontos turísticos mais próximos.
- Monitoramento de abalos sísmicos – abalos distorcem as ondas do GPS, que são percebidas e verificadas.
- Aplicações industriais – podem guiar tratores por plantações para ajudar com a aplicação de pesticidas.
- Uso militar – obtenção de coordenadas de ataque.
- Uso em segurança – monitoramento de trens e caminhões de carga por parte de empresas de logística.

2.6.5 **Infravermelho** (Adams, Schilit, Gold, & Tso, n.d.)

Mantido pela IrDA (*Infrared Data Association*), consiste de ondas eletromagnéticas de radiação operando em frequências entre 300GHz e 400THz.

O uso do infravermelho possui algumas vantagens, como prover a localização exata de objetos, por exemplo, mas que trazem o ônus de exigir que os dispositivos receptores e transmissores estejam alinhados fisicamente para executar a comunicação. Outra limitação do infravermelho é que o tráfego de dados atinge o máximo de 115kbps. O infravermelho, atualmente, permite que uma grande quantidade de aparelhos existentes no mercado troquem informações entre si, tais como celulares, impressoras, computadores, controladores de aparelhos de televisão, etc.

A comunicação através de infravermelho ainda se mostra interessante mesmo com sua conhecida limitação de transferência de informações justamente devido à limitação de alcance da mesma. Como o infravermelho não atravessa paredes, ele se mostra uma forma de comunicação interessante para delimitar perímetros de alcance do dispositivo que estiver operando em nível de um ambiente apenas, como por exemplo, uma sala de conferências em que seja interessante conhecer os dispositivos de todos os participantes.

2.7 Acesso ao hardware

Nesta seção está apresentado o estudo feito sobre a utilização dos hardwares relacionados acima através das APIs do Android. Esta seção foi criada com o propósito de apresentar a viabilidade de uso para alguns dos hardwares apontados pela seção 2.5.

A importância do conhecimento sobre como o SO trata as informações que cada uma das APIs disponibiliza se torna importante para o momento em que elas são obtidas pela aplicação (conforme detalhado na seção 6.3.2).

2.7.1 NFC (Google's Android NFC API, 2011)

O pacote *android.nfc* possui três classes principais, que são responsáveis pela manipulação do adaptador NFC. A API mostrou-se simples de ser utilizada, pois concentra todas as operações necessárias nas classes abaixo:

- NFCManager – usada para obter o controle do adaptador.
- NFCAdapter – representa o adaptador, que é o meio de executar as operações de NFC.
- NDEFMessage – classe que representa a troca de informações entre o dispositivo e as tags NFC.

2.7.2 Bluetooth (Google's Android Bluetooth API, 2011)

O pacote *android.bluetooth* possui 9 classes principais. Estas classes são utilizadas pelas aplicações para se comunicar com o adaptador de Bluetooth e utilizar as funcionalidades que o mesmo proporciona. O uso do Bluetooth em dispositivos Android também se mostrou simples de ser

feito devido à concentração das operações em poucas classes. Um experimento com objetivo de verificar a complexidade de utilização da API do Bluetooth foi feito e está apresentado no item 2.7.2. As classes principais estão listadas abaixo:

- BluetoothAdapter – representa o adaptador e é usada para executar as operações com o bluetooth.
- BluetoothDevice – representa um dispositivo com Bluetooth ativo. É utilizada para manter conexão com outros dispositivos.
- BluetoothSocket – representa uma interface para comunicação entre dispositivos bluetooth através de *Sockets*, que permite leitura e escrita de dados através de InputStream e OutputStrem (similar ao *Socket* de PCs já conhecido).
- BluetoothServerSocket – representa um servidor Socket que está atento a novas conexões. Estabelece conexão através de um objeto BluetoothSocket com novos dispositivos.
- BluetoothClass – classe *read-only* que contém informações sobre o dispositivo Bluetooth acessado.
- BluetoothProfile – classe que determina o perfil de conexão entre dois dispositivos.
- BluetoothHeadset – classe que dá suporte a *headsets* (fones de ouvido com ou sem microfone) para serem usados junto dos dispositivos.
- BluetoothA2dp – determina a qualidade do som que pode ser transferida via *streaming* entre dois dispositivos bluetooth.
- BluetoothProfile.ServiceListener – comunica aos dispositivos quando o bluetooth é conectado ou desconectado do servidor.

2.7.3 IEEE 802.11 (Google's Android WiFi API, 2011)

O pacote *android.net.wifi* provê os meios de estabelecer a comunicação necessária para que o aplicativo consiga obter informações do adaptador wireless, como velocidade da conexão, endereço IP, estado da conexão, etc. Assim como as anteriores, o pacote de wifi mostrou ser simples de ser utilizado pois concentra as operações necessárias em poucas classes. Um experimento também foi feito para demonstrar o potencial e complexidade de uso da API e está detalhado no item 2.7.1. As principais classes estão listadas abaixo:

- ScanResult – informa dados sobre pontos de acesso encontrados.
- WifiConfiguration – classe que representa uma conexão WiFi configurada completamente.
- WifiConfiguration.AuthAlgorithm – possui os algoritmos de autenticação padrão da IEEE 802.11 a serem utilizados para garantir segurança à conexão.
- WifiConfiguration.GroupCipher – grupos de algoritmos de encriptação conhecidos.
- WifiConfiguration.KeyMgmt – classe que possui as senhas configuradas.

- WifiConfiguration.PairwiseCipher – grupos de algoritmos de encriptação conhecidos de WPA.
- WifiConfiguration.Protocol – protocolos de segurança conhecidos.
- WifiConfiguration.Status – classe que contém os possíveis status da configuração.
- WifiInfo – contém o estado de todas as conexões wifi ativas ou em processo de configuração.
- WifiManager – classe que controla todas as operações possíveis com uma conexão Wi-Fi.
- WifiManager.MulticastLock – classe que permite a um aplicativo receber pacotes multicast (de vários locais).
- WifiManager.WifiLock – classe que permite aos aplicativos manter o adaptador Wi-Fi sempre ativo.
- WpsInfo – classe que representa a configuração de um Wi-Fi protegido.

2.7.4 GPS (Google's Android Location API, 2011)

As classes do pacote *android.location* permitem que os aplicativos obtenham informações sobre o adaptador GPS do aparelho. Através delas é possível identificar a localização do usuário através dos satélites GPS. Abaixo está a descrição sobre as classes deste pacote e suas utilidades:

- Address – classe que representa um endereço, como por exemplo um conjunto de palavras descrevendo uma localização.
- Criteria – classe que representa critérios para seleção de um provedor de localização.
- Geocoder – classe para manipular códigos geográficos.
- GpsSatellite – representa o estado de um satélite GPS.
- GpsStatus – representa o estado do adaptador GPS.
- Location – classe que representa uma localização geográfica obtida em um momento específico.
- LocationManager – provê controle e acesso aos serviços de localização.

2.7.5 Infravermelho

O Android atualmente não possui API para suporte ao infravermelho.

2.7.6 Acelerômetro

O acelerômetro no Android não possui um pacote específico para agrupar suas classes. Possui apenas três classes ligadas diretamente aos eventos do mesmo. Abaixo está uma breve descrição sobre cada uma delas:

- Sensor – classe que possui as constantes para serem utilizadas para obter dados sobre cada sensor disponível no aparelho através da classe SensorManager. Esta classe agrupa as constantes de vários tipos de sensores, tais como acelerômetro, giroscópio, pressão, orientação, etc.
- SensorManager – provê controle e acesso ao serviço do acelerômetro, dando acesso a métodos que informam inclinação do aparelho, orientação, rotação, mudança de ângulo, etc.
- SensorEvent – classe que é obtida pela SensorManager para informar a aplicação que alguma alteração foi feita no estado do sensor. Esta classe fornece informações sobre mudanças no ambiente e nas informações captadas pelo sensor.

2.8 Experimentos

Para as APIs de WiFi e Bluetooth, um pequeno aplicativo foi montado a fim de explorar os métodos principais de cada uma delas. Os experimentos foram realizados para identificar a complexidade de uso de cada um dos hardwares estudados e então definir sua aplicabilidade no projeto. Para ambos os testes, foi feita uma ativação do dispositivo e um scan completo na área afim de obter os dispositivos ligados. A API de NFC não foi testada por falta de recursos de hardware para executá-la, e a de GPS e acelerômetro não foram concluídas por falhas no hardware disponível. Abaixo seguem os exemplos montados e os resultados obtidos:

2.8.1 WiFi

Para a API de WiFi, as classes de gerenciamento foram utilizadas com propósito de varrer a área em uso e obter as redes disponíveis. A figura 5 apresenta o teste de utilização do detector de redes wireless.

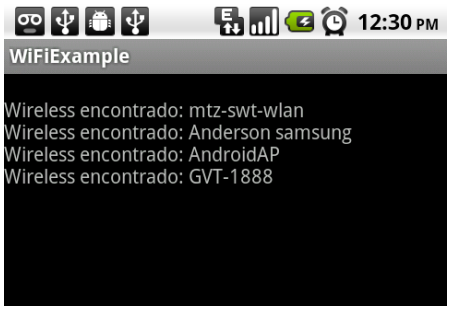


Figura 5 - Teste de obtenção de redes disponíveis – criado pelo autor

O resultado obtido foi a lista de redes disponíveis no momento em que o teste foi executado.

2.8.2 Bluetooth

Para a API de Bluetooth o mesmo teste foi realizado. As classes de gerenciamento do adaptador foram utilizadas para obter a lista de dispositivos disponíveis ao alcance, e uma tela

com a descrição dos mesmos e seus MAC address foi gerada. A figura 6 apresenta o resultado obtido com o teste:

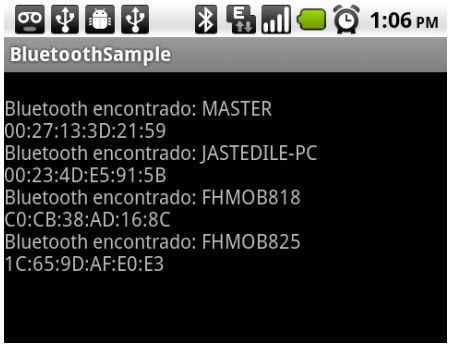


Figura 6 - Teste de obtenção de redes bluetooth disponíveis – criado pelo autor

3. Trabalhos relacionados

Na área de sensibilidade móvel a contextos há iniciativas já finalizadas, e outras também em desenvolvimento, tanto de pesquisadores como de organizações privadas. Abaixo estão apresentadas algumas que usam o conceito de sensibilidade a contextos e que também, de alguma maneira, interferem no desktop do usuário e já utilizam tecnologias ou conceitos que foram empregados neste projeto.

Estes projetos vão desde simples iniciativas científicas, com objetivo apenas de demonstrar a tecnologia e de realizar pesquisa, até produtos com objetivo fundamentalmente comercial e que atrairia certo perfil de consumidores. Junto disso, as iniciativas descritas também darão exemplos de aplicações que alteram a maneira como o usuário interage com seu desktop. Estes diferentes trabalhos foram escolhidos por mostrarem diferentes abordagens a alterações no desktop de dispositivos móveis sempre tendo como início algum processo adaptável a contextos.

Nesta seção estão apresentados, em ordem cronológica, sete diferentes trabalhos relacionados à sensibilidade a contextos:

- **Softwares com sensibilidade a contextos e pouca alteração no desktop do usuário (1995 a 2009):** são trabalhos genéricos e outros direcionados a determinadas áreas que tratam a sensibilidade a contextos com uma abordagem comercial.
- **Framework genérico em detalhe com alteração no desktop variável de acordo com o desenvolvedor (2009):** trata-se de uma proposta semelhante à mostrada por este programa (com detalhes na seção 5.1) explicada em mais detalhes.
- **Otimização de carregamento de aplicativos (2011-2012):** uma visão sobre uma proposta comercial da Microsoft sobre a sensibilidade a contextos aplicada diretamente no que não vemos do desktop do sistema operacional e não a nível de aplicativos.
- **Softwares de desktops adaptáveis (2011, 2012 e 2013):** softwares disponíveis comercialmente com objetivo de alterar a maneira como o usuário interage com seu desktop.

Após a identificação destes trabalhos foi criada uma tabela comparativa para ilustrar áreas que são abrangidas ou não pelos mesmos, que pode ser vista no item 3.5.

3.1 Softwares com sensibilidade a contextos (1995 a 2009)

Abaixo estão descritas três iniciativas encontradas e identificadas como bases para a criação de aplicações sensíveis a contextos. O estudo sobre estas iniciativas se motivou pelo fato de utilizarem dados de localização do usuário em seus projetos e de eventualmente sinalizarem dificuldades e principalmente experiências adquiridas com os anos de pesquisa na área.

3.1.1 **O ParcTab** (Want et al., 1995)

Criado pela Xerox em 1995 foi um dos primórdios no conceito de sensibilidade a contextos para dispositivos móveis. Consistia de aparelhos hand-held customizados chamados ParcTabs, que possuíam uma ferramenta criada utilizando comunicação através de infra-vermelho. O infra-vermelho foi escolhido por sua característica de não poder atravessar paredes, e desta maneira limitar o espaço de tráfego de informações a uma sala, por exemplo (que pode caracterizar um contexto). Com esta plataforma algumas aplicações foram criadas para passar informações de localização e pequenos lembretes aos usuários, desde que fossem aplicáveis ao contexto em questão.

3.1.2 **O trabalho de** (Dey, Abowd, & Salber, 2001)

Descrevia um framework e um toolkit para suportar uma rápida prototipação de aplicações sensíveis a contextos. O framework consistia essencialmente de um conjunto de manipuladores de sensores, que notavam alterações no ambiente e enviavam estas informações aos widgets conhecidos. Apresentou um conceito de widgets que possuíam uma interface comum e que eram capazes de alterar seu comportamento de acordo com as alterações em dados dos sensores e obter informações como o histórico do contexto. Além disto, apresentava uma nova forma para facilitar a interpretação e utilização das saídas do widget, que tinha a interação com o usuário como principal foco.

3.1.3 **Otimização de ambiente residencial** (Sang-Hak & Tae-Choong, 2004)

O trabalho sobre ambiente residencial consiste de uma série de adaptações feitas em diferentes dispositivos existentes em residências. Foram propostas diversas adaptações e o trabalho foi dividido em diferentes tipos de serviços que foram explorados. A escolha dos serviços investigados neste trabalho foi feita com base no potencial para comercialização, usabilidade, relevância e também dificuldade de implementação. Os serviços escolhidos pelos autores foram:

a) **Serviço de mídia:** o serviço de mídia criado foi pensado para utilizar fotos e/ou vídeos e poderia ser implementado na câmera que captura imagens e vídeos e em paralelo na televisão do usuário. Nos locais onde o usuário capturasse quaisquer imagens – como museus, shoppings, praças, etc –, o próprio dispositivo poderia identificar tags como local, horário e informações sobre o ambiente. Então a partir do momento em que as mídias fossem visualizadas em um computador/televisão/etc, poderia ser feita uma interpretação das informações capturadas e oferecer opções de enviar as imagens às pessoas identificadas nas fotos (através de identificação de faces e comparação com amigos de redes sociais, por exemplo) juntamente com informações sobre o local em que estiveram, ou ainda exibir links e convites para ler mais sobre os locais que foram visitados e/ou artefatos vistos.

b) **Serviço de saúde:** com o crescimento da população e da preocupação com qualidade de vida, alguns serviços mais básicos de saúde começam a se tornar interessantes para um número mais expressivo de usuários. Pensando nisto, as pessoas poderiam possuir algum hardware previamente preparado em suas residências, que utilizariam diariamente para checar informações básicas como peso, gordura corporal, pressão sanguínea, batimentos cardíacos, entre outros. Estes dados poderiam ser enviados em tempo real a um médico e/ou a um *personal trainer*. O médico poderia indicar ao paciente uma alimentação saudável para aquela época do ano, enquanto o *personal trainer* poderia indicar exercícios que o usuário deveria realizar para atingir algum objetivo em específico.

c) **Serviço de controle de ambientes:** o serviço de controle de ambientes foi pensado para ser utilizado em diferentes circunstâncias e em vários dispositivos, desde que gerenciados por um controlador central. Para um dormitório, por exemplo, alguns sensores podem ser instalados de maneira a prover uma boa noite de sono através de ajustes automáticos de ar condicionado, humidificador de ar e ainda músicas. Já para a residência como um todo, sensores podem ser instalados para identificar quando algum morador específico está se dirigindo para a casa e previamente iniciar sua climatização, abrir janelas, ligar aparelhagem de som e etc, conforme o gosto do morador.

Para o exemplo da seção 3.1.4, em específico, foram apresentadas algumas dificuldades e motivações que levaram à suas propostas devido a sua simplicidade e semelhança com produtos conhecidos atualmente. Também foram mostradas as tecnologias requeridas, tanto de hardware como software, o que fez evidente que todas elas podem ser implementadas com as tecnologias atuais, demandando apenas tempo para a criação dos hardwares e softwares.

3.1.4 O framework criado por (Johnson, 2007)

Semelhante ao framework descrito na seção 3.1.2, porém mais avançado tecnologicamente e com foco em aparelhos operando com *Windows mobile 5* e utilizando a tecnologia *.Net Compact Framework*, procura abordar dois pontos relevantes da área: apresenta uma plataforma com objetivo de que terceiros possam desenvolver rapidamente aplicativos sensíveis a contextos e procura entender limites às aplicações sensíveis a contextos por limitações tecnológicas e/ou de aparelhos.

3.1.5 Buscas móveis sensíveis a contextos (Gui et al., 2009)

Este trabalho foi descrito em sete sessões, que explicam os módulos do sistema. O foco deste trabalho está em interagir com o usuário através das buscas feitas em dispositivos móveis. Os itens de “A” a “E” descrevem módulos da aplicação de buscas que operaria em dispositivos móveis. Já o item “F” logo abaixo possui a explicação de como é feita a ligação entre todas as informações coletadas e um exemplo de sua aplicabilidade:

a) **Interpretador de contextos:** responsável por interagir com sensores coletando informações relativas ao contexto e resolver problemas relativos às detecções de contextos. Em adição a isso, o interpretador também recebe informações dos dispositivos móveis para guardar dados sobre os usuários da aplicação.

b) **Registro de serviços:** o registro de serviços interage com aplicações de terceiros para possibilitar o registro de aplicações que os dispositivos móveis utilizarão. Como exemplo há a interferência de uma marca de comidas qualquer, que pode oferecer seu serviço ao detectar que o usuário está comprando de uma concorrente.

c) **Interpretador de serviços:** o interpretador de serviços interage com o aparelho móvel para interpretar as requisições dos serviços pelo usuário.

d) **Gerenciador de contexto:** o gerenciador de contextos formata e processa as requisições. Caso seja possível executar a requisição com os serviços já disponíveis, o gerenciador executa e envia os dados de volta aos dispositivos móveis. Caso não seja possível, a requisição é repassada ao gerenciador de buscas sensíveis a contextos.

e) **Registro de protocolos:** o registro de protocolos é o responsável por registrar e armazenar os dados de comunicação com os protocolos utilizados, como TCP/IP, WLAN, GSM, CDMA 2000, WCDMA ou TD-SCDMA. O armazenamento destes tipos de comunicação é importante para garantir que todos os tipos de requisições serão recebidas e interpretadas.

f) **Gerenciador de buscas sensíveis a contextos:** o gerenciador de buscas sensíveis a contextos, principal módulo desta aplicação, é responsável por fazer buscas levando em consideração os dados do contexto onde o usuário está inserido. Ele deverá reunir as informações identificadas pelos módulos descritos anteriormente, de maneira a levá-los em consideração quando o usuário realizar uma busca. Um exemplo trivial seria que uma dona de casa americana e um usuário de *iPhone* terão opiniões e comentários divergentes sobre a palavra *apple*. Desta maneira, cabe ao gerenciador buscar as informações relativas ao contexto em que o usuário está inserido e não apenas buscas estáticas.

3.2 Framework genérico em detalhe por (Kovács, Mátételki, & Pataki, 2009):

O framework proposto por (Kovács et al., 2009) descreve um modelo para aplicações sensíveis a contextos através de uma arquitetura (figura 7) muito semelhante com a idealizada e motivadora da criação deste programa. O trabalho foi estudado devido à semelhança com a arquitetura pensada para a criação deste ambiente. As semelhanças encontradas se devem ao fato de o mesmo usar recomendações e também ontologias para trafegar dados entre os diferentes módulos que o compõem. As semelhanças são minimizadas ao passo que a aplicação estudada foca apenas a criação de um framework, e a apresentada por este programa desenvolveu uma ferramenta por completo. A

semelhança poderá ser vista durante a descrição do programa e com sua figura de ilustração (seção 5.1):

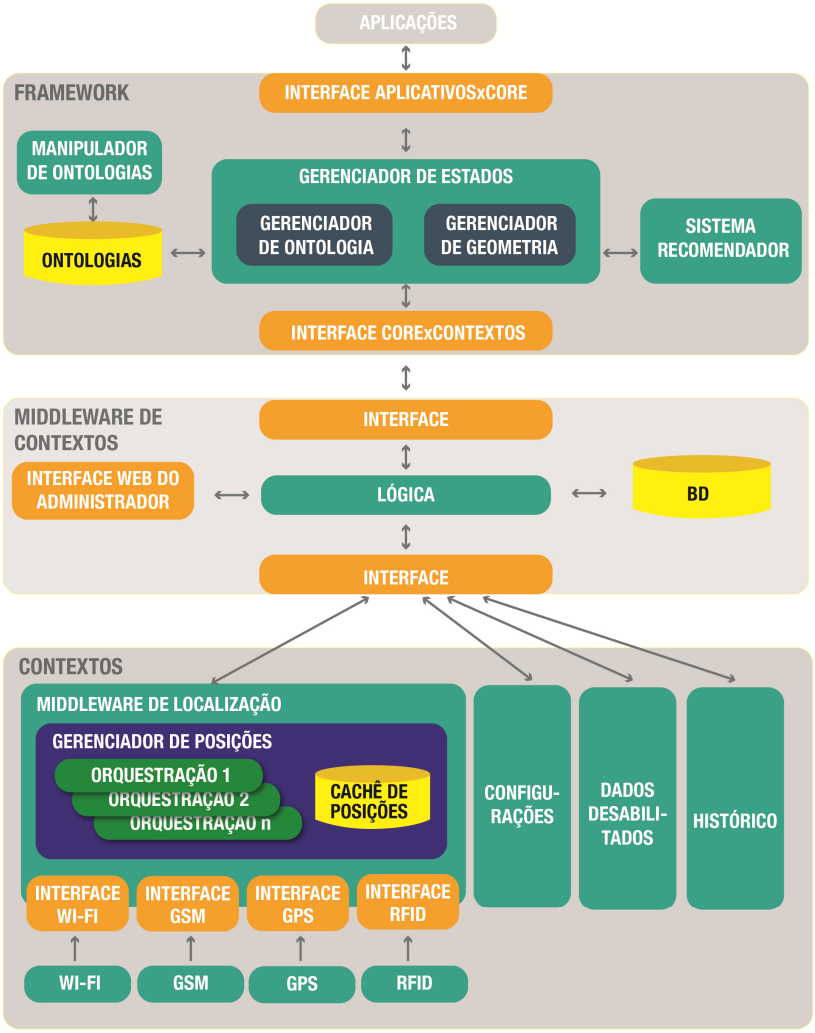


Figura 7 - Framework proposto pelo trabalho relacionado. Fonte: (Kovács et al., 2009)

O framework mostrado na figura acima descreve a comunicação da seguinte maneira:

- As aplicações comunicam-se com o framework através de um gerenciador de ontologias (*ontology manager*) e de dados de localização (*geometry manager*). Estes dados de localização são utilizados para calibrar um sistema de recomendações (*recommender system*), já os dados de ontologias são armazenados em uma base (*ontology*) e manipulados por um handler (*ontology handler*).
- Os gerenciadores de ontologia e de localização se comunicam com o middleware dos contextos (através do *Context2Core WS Interface*), que possui lógica (*Logic*) para receber os serviços, com um banco de dados (*DB*) que guardará estas informações, e com a interface de administração web (*Admin web interface*).
- O middleware de contextos por sua vez faz a comunicação com os contextos, que podem ser identificados através de várias maneiras: localização (utilizando wifi (*WiFi*), GSM (*GSM*), GPS

(*GPS*) e RFID (*RFID*)), preferências explicitadas pelo usuário (*preference*), histórico (*history*) e inclusive uma eventual incapacidade de obter alguns dos dados e/ou informações (*disability*).

Apresentado através de uma figura diferente da arquitetura proposta para este programa, as semelhanças encontradas foram:

- Utilização de ontologias para descrever os serviços existentes na aplicação.
- Utilização de agentes (descritos na figura acima como managers) para troca de informações.
- Utilização de um armazenamento para guardar os dados coletados.
- Criação de recomendações para os aplicativos.

Já algumas das diferenças seguem destacadas abaixo:

- Ausência de um módulo de interface com o usuário por parte do trabalho estudado (visto que não era o foco do mesmo).
- Ausência do conceito de guardar os dados de histórico de uso na nuvem (o framework propunha apenas um BD (banco de dados)).
- Determinação, por parte do projeto estudado, de quais hardwares serão obrigatoriamente utilizados. O ambiente descrito na seção 5 prevê o uso de vários tipos de hardware, variando de acordo com sua disponibilidade em cada dispositivo.
- Diferença entre a recomendação de serviços (do projeto estudado) e a recomendação de alterações no desktop semântico por parte da proposta apresentada.

3.3 Otimização de carregamento de aplicativos (2012)

Com (Chu et al., 2012), a Microsoft iniciou uma pesquisa em 2011 que mencionava a importância da interferência do sistema operacional em aplicativos e outros serviços. Na figura 8 abaixo podemos ver a visão de quais serviços do sistema poderiam ser afetados pelo contexto identificado pelo sistema operacional.

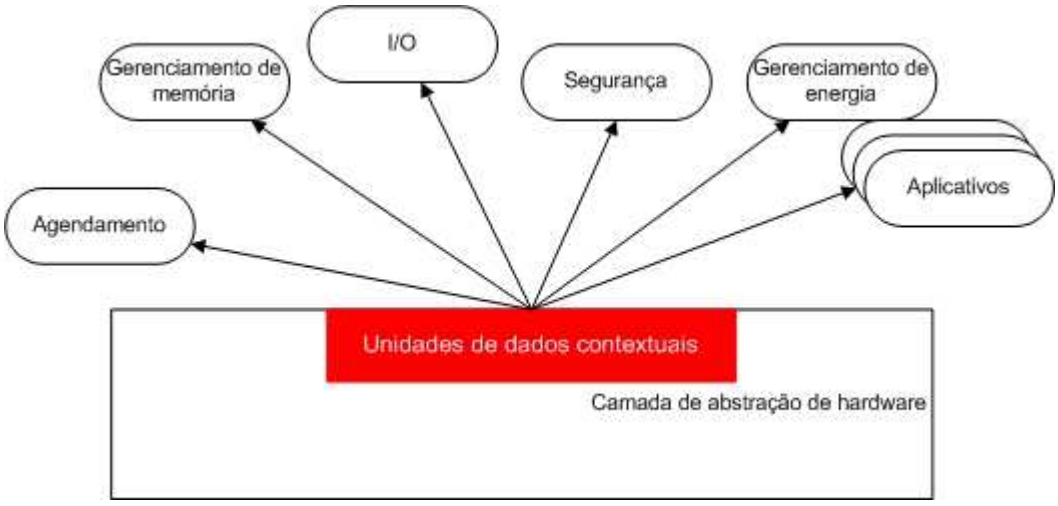


Figura 8 - Serviços que poderiam ser afetados pela identificação de contexto pelo sistema operacional. Fonte: (Chu et al., 2012)

Abaixo está descrita uma breve visão com mais detalhes de como o contexto identificado pelo SO poderia afetar positivamente cada uma das áreas apontadas na figura 8:

- **Gerenciamento de memória:** o gerenciamento da memória ainda é complexo em smartphones principalmente devido a aberturas de aplicativos cada vez mais rápida sendo exigidos pelos usuários. A detecção de contextos poderia dar dicas ao sistema operacional sobre quando fazer um "pré carregamento" de um determinado aplicativo para então economizar espaço em memória.
- **Segurança:** o dispositivo poderia identificar, por exemplo, que o usuário está em sua casa e então desativar o travamento por senha.
- **Agendamentos:** o agendamento de tarefas, na maioria dos sistemas operacionais móveis atuais, acontece através de tarefas sendo executadas em background, que têm o objetivo de disparar eventos quando determinado horário é atingido e/ou alguma ação é executada. Com a detecção do contexto, o agendamento de tarefas poderia ser otimizado para que os eventos que são executados apenas em situações específicas não exijam cuidados da tarefa principal, economizando assim memória, processamento e consequentemente tempo de bateria.
- **Gerenciamento de energia:** com o auxílio de contextos o dispositivo conseguiria, por exemplo, alertar o usuário de que, ao acordar pela manhã, não haveria bateria suficiente para uso durante o restante do dia e informar que ele deve levar um carregador consigo.
- **I/O:** para informações de entrada e saída do smartphone, poderia ser aplicado, por exemplo, para que durante uma reunião o volume do toque do aparelho fosse ajustado automaticamente para algo próximo do mínimo, ao mesmo tempo que para durante uma festa, ele seja ajustado para o máximo. Assim o volume do aparelho estaria adequado aos dois ambientes sem exigir intervenção do usuário.

O trabalho da Microsoft apresentado torna-se pertinente a este projeto por tratar de assuntos de forte interesse do usuário final, e que poderiam ser facilmente tratados por um aplicativo de *home replacement* (seção 5.3.2) sem interferir diretamente no sistema operacional. Mostra-se como uma possível extensão do ambiente apresentado neste trabalho, mas que não foram incluídos nesta etapa do processo devido ao tempo envolvido.

3.4 Softwares de desktops adaptáveis (2010, 2011 e 2012)

Foram escolhidas três aproximações a dekstops adaptativos inteligentes já disponíveis comercialmente. Dois destes desenvolvimentos são voltados à plataforma Android e mostram algumas maneiras que outros desenvolvedores encontraram para alterar e configurar o desktop. A outra aproximação encontrada é voltada ao desktop de computadores operando com Windows, que foi incluída para pesquisar diferentes tentativas de adaptação de desktops que outras plataformas executam.

No Android, os aplicativos que substituem a *home* do smartphone são chamados “Launchers”, pois são responsáveis pela interface que vemos quando não estamos usando nenhum aplicativo. São usados para executar (em inglês, *launch*) os novos aplicativos que o usuário deseje.

Estas três ferramentas foram escolhidas devido a sua relevância em seu meio e arrojo em termos de complexidade. Todas as ferramentas encontradas se mostraram líderes, ou próximas disso, em seus segmentos de negócio (Falcon, 2012; Vasile, 2011; Whitwam, 2011), porém nenhuma delas mostra tentativas de dinamizar seu funcionamento, seja através de sensores ou mesmo através de um simples registro de uso. São elas:

3.4.1 **Fences** (Starck, 2010)

O Fences, da Stardock, é um software que possibilita ao usuário de Windows (possui edições para todas as versões do Windows superiores ou iguais à XP) organizar seu desktop. Pode esconder ícones através de um simples comando e também organizá-los em um grupo chamado “Fence” (conforme mostrado pela figura 9). Estes grupos podem ser facilmente movidos e também customizados. Podem ter sua cor de fundo alterada, rótulos de identificação alterados, e serem escondidos ou mostrados.

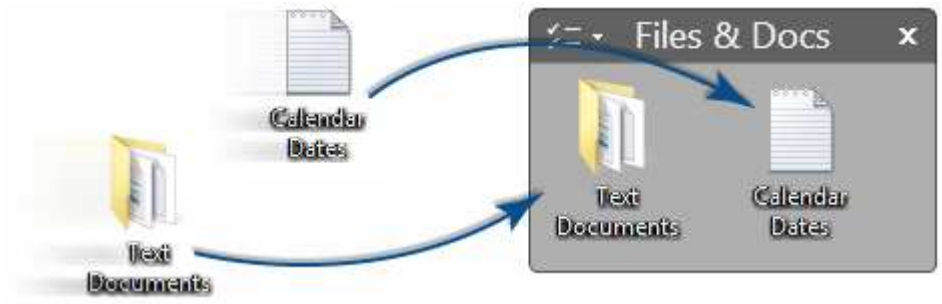


Figura 9 - Exemplo de agrupamento de ícones no Fences. Fonte: (Starck, 2010)

O Fences ainda permite a criação de perfis de uso (ou, como são chamados no software, “páginas”), ou seja, condicionar seu funcionamento à atividade que o usuário estiver praticando. Caso o usuário use o mesmo computador tanto para trabalho quanto para diversão, poderá criar dois perfis e então, através de um simples comando, fazer com que os atalhos que usa para o trabalho sejam substituídos pelos atalhos que usa mais comumente em casa para outras atividades. Esta transição entre perfis exige um comando do usuário e o software não provê maneiras de dinamizar esta troca até sua versão atual.

Abaixo a figura 10 apresenta um desktop alterado com uso do Fences:

Figura 10 - Desktop alterado com uso do Fences. Fonte: (Starck, 2010)

O Fences se mostra como uma boa ferramenta para organizar tarefas simples do desktop e que podem trazer incomodo a um determinado grupo de usuários. Entretanto não mostra nenhuma tentativa de propor organizações diferentes no desktop, nem de otimizar a troca de perfis, por exemplo. É uma ferramenta de excelente qualidade dentro de sua proposta, mas ainda estática e totalmente dependente do usuário.

3.4.2 **LauncherPro** (Pplware, 2011)

O LauncherPro, cuja dona é uma empresa de mesmo nome, é um dos mais conhecidos launchers customizados para Android (Guiss, 2011). Permite ao usuário trocar o estilo do dock (barra de ícones na parte inferior da tela) para até três vezes seu tamanho original, inclui vários efeitos adicionais de transição de telas (que são usados para criar uma boa experiência de uso), permite criar atalhos para funções específicas de aplicativos (como ligar/desligar wireless, bluetooth, etc), e também inclui alguns widgets próprios que não existem livremente no Android Market, tais como widgets para Facebook, Twitter, Contatos e Gmail.

O LauncherPro é exclusivo para Android e possui versões compatíveis com o sistema operacional desde sua versão 2.0. A figura 11, abaixo, mostra o funcionamento de duas características importantes da ferramenta:

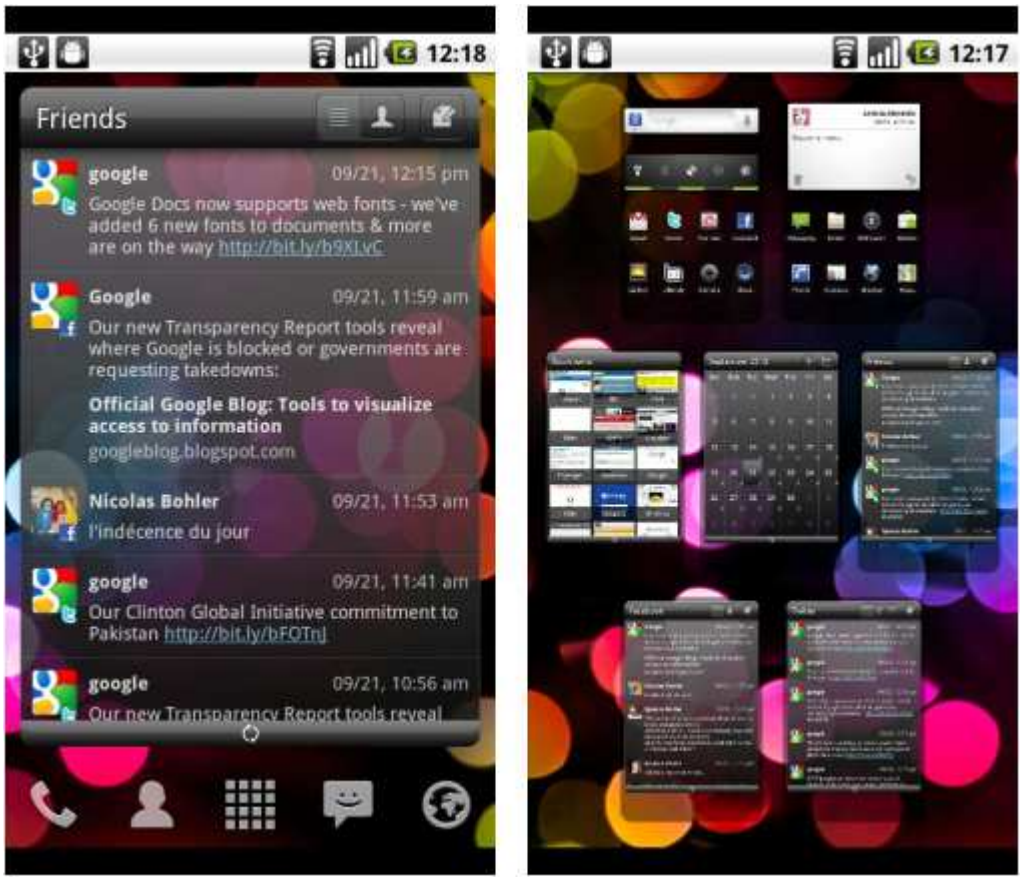


Figura 11 - Figura do LauncherPro em uso. À esquerda um widget customizado que mostra atividades de amigos. À direita, uma funcionalidade que mostra todas as telas do aplicativo em uma só. Fonte: LauncherPro

O LauncherPro é um ótimo aplicativo para substituição do desktop original do Android, pois seu funcionamento é muito mais veloz e pelo fato de trazer várias outras funcionalidades junto disto. Porém ele também não demonstra nenhuma habilidade em alterar seu funcionamento dinamicamente. Alterações em seu comportamento são sempre feitas pelo usuário e o aplicativo não interfere nas mesmas.

3.4.3 **ADWLauncher** (Daquino, 2012)

O ADWLauncher, da ADWThings, é um launcher de simples configuração e muito rico em possibilidades. Assim como o LauncherPro, permite diversas alterações na home do smartphone, tendo as principais relacionadas abaixo:

- Alterar o funcionamento da barra de status para mostrar ou ocultar alertas configurados pelo usuário.
- Definir os tamanhos dos ícones que são mostrados ao usuário, permitindo um maior número de ícones na tela, por exemplo.
- Definir efeitos para a transição entre as diferentes telas da área de trabalho.
- Suporte para notificações sobre os ícones da home, tornando o acesso a informações mais fácil.

A figura 12 mostra uma de suas mais notáveis funcionalidades, que é a criação de ícones para atividades específicas dentro de aplicativos.

Figura 12 - Gerenciamento de ícones criados para funções existentes dentro de aplicativos. Fonte: ADWLauncher

Este segundo trabalho, diferentemente do primeiro visto na seção 3.4.2, possui um desenvolvimento muito mais complexo sobre suas funcionalidades e configurações possíveis. Possui muitas possibilidades de alteração da home para fazê-la ficar da maneira como o usuário deseja interagir com seus aplicativos.

Igualmente aos outros dois trabalhos sobre desktops adaptáveis (seções 3.4.1 e 3.4.2), o ADW Launcher não foi desenvolvido com intenção de dar algum tipo de inteligência ao funcionamento. Todas as alterações que o mesmo proporciona são estáticas e não interagem com informações geradas pelo usuário.

3.5 Comparação entre trabalhos pesquisados

Com os trabalhos apresentados acima, vários aspectos do campo de sensibilidades a contextos puderam ser vistos.

- No item 3.1 três iniciativas permitiram ver que o foco de pesquisadores, por algumas vezes, foi permitir que desenvolvedores utilizassem suas ferramentas como um meio para controlar os sensores, e preocupar-se apenas com o desenvolvimento da interface e comportamento da

aplicação. Esta seção foi usada para entender sobre localização e uso da mesma para criação de contextos e adaptação da interface.

- O item 3.2 foi responsável por apresentar uma proposta semelhante à prevista neste trabalho, que foi utilizada para pesquisa e também para procurar diferenças entre trabalhos já existentes.
- No item 3.3, outro tipo de iniciativa, com foco comercial, foi mostrada para exemplificar outros tipos de trabalhos na área e evidenciar que a exploração de sensibilidade a contextos interessará fortemente fontes comerciais, desta maneira dando um destaque importante ao projeto apresentado neste trabalho.
- O item 3.4 mostrou três abordagens diferentes para um mesmo problem. As modificações possíveis nos desktops do Android e do Windows foram analisados para dar conhecimento sobre as alternativas possíveis para o desenvolvimento do projeto.

Para abordar a comparação entre os artigos pesquisados e destacar a diferença entre os mesmos e o ambiente mostrado neste trabalho, a tabela 4, mostrada abaixo, foi criada. Os critérios escolhidos foram eleitos de maneira a diferenciar a abordagem de cada um deles e mostrar que possuem fraquezas e diferenças com relação ao trabalho desenvolvido.

Com a tabela mostrada abaixo, a contribuição com relação à interface de dispositivos torna-se mais clara. É esperado dar mais rapidez ao uso de um dispositivo móvel através das modificações em seu desktop e assim facilitar o uso por parte do usuário final.

Os trabalhos pesquisados foram encontrados relacionando contextos com os dados que são mostrados ao usuário (ícones, alertas, etc). Cada abordagem possui características diferentes para definir a interface do usuário, que é o foco deste trabalho, e por isso os seguintes critérios foram escolhidos:

1Interfere no desktop – indica se o trabalho relacionado executa qualquer ação que interfere de alguma maneira no desktop que o usuário já está acostumado a usar.

2Substitui o desktop – indica se o trabalho relacionado substitui o desktop do usuário como um todo, e não apenas altera alguns pontos para fins de visualização.

3Organiza o desktop – indica se o trabalho relacionado altera a forma como os dados com os quais o usuário interage normalmente são apresentados.

4Otimiza o desktop – indica se o trabalho relacionado acelera ou facilita de alguma maneira processos que o usuário esteja acostumado a realizar.

5Recomenda usos no desktop – indica se o trabalho relacionado indica ou sugere quaisquer tipos de processos no desktop do usuário.

6Apresenta economia de bateria – indica se o trabalho relacionado apresenta alguma economia no consumo de bateria.

	1	2	3	4	5	6
Parctab		X				
Widgets	X	X		X		
Ambiente residencial			X			
Framework para Windows Phone	X		X			
Buscas Móveis	X					
Framework genérico				X		
Otimização de carregamento	X	X		X		
Fences	X		X			
LauncherPro	X	X	X	X		
ADWLauncher	X	X	X			

Tabela 4 – Comparação entre trabalhos relacionados

Com a tabela acima podemos ver que nenhum dos aplicativos mostrados pelo levantamento bibliográfico possuem características de recomendação de uso ou de economia de bateria. Estes pontos foram focados pelo trabalho para diferenciá-lo dos restantes e apresentar melhorias na experiência do usuário.

4. MyFace

Este trabalho de dissertação está inserido em um programa, de mesmo nome, composto por três projetos. Antes da apresentação do mesmo, foi feita uma contextualização sobre o programa como um todo, onde alguns termos usados na explicação do projeto estão apresentados e descritos.

4.1 Visão geral do programa MyFace

Este projeto está sendo desenvolvido em paralelo a outros dois projetos. Este programa está sob coordenação do professor Dr. Sérgio Crespo e está sendo desenvolvido junto ao programa de mestrados do PIPCA (Programa Interdisciplinar de Pós-graduação em Computação Aplicada - Unisinos). Abaixo, na figura 13, está apresentada uma visão macro sobre a integração entre os três projetos: CONTEXT SERVER, CONTEXT MANAGER, e INTERFACE SENSÍVEL AO CONTEXTO, sendo que este último é tratado por este trabalho.

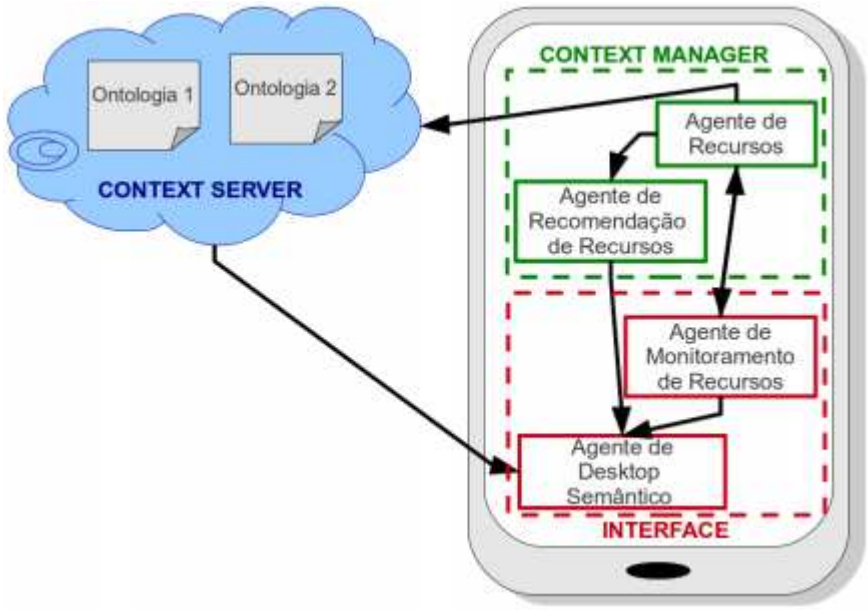


Figura 13 - Arquitetura macro do programa – criado pelo autor

O modelo da figura 12 contém os seguintes componentes:

- **Dispositivo móvel** sendo utilizado pelos usuários contendo os projetos INTERFACE e CONTEXT MANAGER, formados pelos agentes de software descritos abaixo:

  a) **agente de monitoramento de recursos:** monitora as atividades do usuário. Avisa o agente de recursos sempre que uma nova aplicação estiver sendo inicializada e recebe informações de localização (dados de GPS, bluetooth, acelerômetro, etc) e tempo como resposta. Após a troca de informações, o agente de monitoramento de recursos envia para o Context Server o conjunto formado pelo recurso em execução, localização e tempo.

b) **agente de recursos:** recebe a informação de que uma aplicação está em uso e adiciona dados de espaço e tempo, (informações de localização geográfica em conjunto com deslocamento *in-door* através do uso de acelerômetro, juntamente com informações de data e hora), formando assim o conteúdo de uma ontologia de armazenamento do contexto de utilização de um determinado aplicativo no dispositivo móvel. Enquanto o sistema identificar que existe conexão com a internet, a cada alteração armazenada na ontologia, esta será enviada para outro sistema centralizador de informações de contexto do respectivo usuário deste dispositivo móvel, contido no CONTEXT SERVER.

c) **agente de recomendação de recursos:** este agente identifica uma mudança nos dados de uso do usuário gerenciada pelo agente de recursos ou, uma mudança no espaço ou tempo e faz sugestões de uso de recursos para o agente de desktop semântico. As informações enviadas por este agente contêm dados de localização do usuário, tempo e lista de recursos sugeridos; Esta recomendação será formada a partir do histórico de uso de aplicativos já registrado para aquele usuário. Baseado em comportamentos previamente mapeados, as recomendações tenderão a seguir um padrão semelhante às atividades que o usuário tradicionalmente executaria.

d) **agente de desktop semântico:** este agente analisa informações recebidas do agente de recomendação e do agente de monitoramento de recursos para gerenciar a atualização do desktop e dos recursos baseados em seu contexto lógico, ou seja, tendo conhecimento sobre o comportamento previamente armazenado, sobre o ambiente atual do usuário e de informações de tempo e espaço, o aplicativo terá condições de alterar o desktop do smartphone de maneira a deixar evidente o ícone de um aplicativo qualquer, como e-mail por exemplo.

- **Nuvem** (projeto identificado como CONTEXT SERVER): armazena diversas ontologias sobre contextos lógicos no uso de tecnologias e recursos de cada usuário cadastrado que comporão uma federação de ontologias. A nuvem será responsável por armazenar todo o histórico de comportamento dos usuários, juntamente com descrições sobre sua localização geográfica e de tempo, que ajudarão a manter histórico do contexto em que as ações foram tomadas.

4.2 Projetos CONTEXT MANAGER e CONTEXT SERVER

Com esta seção, objetiva-se descrever resumidamente quais dos recursos apresentados acima são abordados pelos outros trabalhos do programa e utilizados por este projeto.

Os dois projetos descritos nesta seção mostram como os dados gerados pelo usuário são armazenados para então serem usados nas recomendações. Para tanto, dois importantes conceitos no desenvolvimento foram definidos:

- **O comportamento armazenado** pode ser um conjunto de quaisquer dados de localização, desde que incluindo sempre o nome do aplicativo executado e algum dado sobre sua localização (implementação detalhada no item 6.3.2.2). Esta informação, ao ser agrupada com o histórico do mesmo usuário, servirá como insumo para gerar as recomendações importantes ao usuário.
- **O histórico criado** será composto de vários pacotes de informações, enviados a cada momento em que o usuário disparar um novo aplicativo no desktop (também descritos no item 6.3.2.1), e que agrupados serão usados para gerar as recomendações pelo Context Manager. Estes dados de histórico serão mantidos na nuvem e guardados pelo projeto Context Server.

O projeto CONTEXT MANAGER pode ser descrito, rapidamente, como:

- Responsável por utilizar os sensores existentes em cada aparelho (descritos pela seção de hardwares estudados – 2.5), identificar as ações que o usuário toma e guardar estes dados, através do Context Server, como histórico para fazer recomendações.
- A partir do histórico criado (e armazenado no CONTEXT SERVER), recomendar alterações no desktop de cada aparelho.

Já o projeto CONTEXT SERVER pode ser descrito, rapidamente, como:

- Responsável por receber os dados de uso do usuário como: sensores, aplicações executadas, horário de atividade, etc. e guardá-los para poder prover dados ao CONTEXT MANAGER para então gerar recomendações.

4.3 A proposta – projeto INTERFACE SENSÍVEL A CONTEXTOS: MyFace

O projeto MyFace desenvolve uma arquitetura para reorganização da interface do desktop do dispositivo móvel. Com isto espera-se facilitar o acesso do usuário a processos e/ou aplicativos que ele comumente utilize. A arquitetura permite a adequação da interface que o usuário visualiza conforme seu contexto lógico, de maneira a facilitar (deixar evidente e disponível), o acesso a aplicativos pertinentes ao mesmo em determinado momento, que serão identificadas através do perfil do usuário.

Para a viabilização da arquitetura, o protótipo “MyFace” foi desenvolvido, e é responsável por gerenciar todo o processo de customização da interface do desktop com o usuário.

Para tanto, o sistema deverá receber informações de outro sistema do smartphone, também inserido neste programa (detalhes nas sessões 5.1 e 5.2), que chamamos CONTEXT MANAGER. Estas informações deverão vir em forma de recomendações, que serão responsáveis por disparar as alterações no desktop do smartphone. A recomendação de aplicativos a ser enviada pelo CONTEXT

MANAGER virá por meio de uma análise de ações especializadas[1] tomadas pelo usuário. O registro histórico de ações especializadas será obtido através do terceiro sistema do projeto, chamado CONTEXT SERVER (detalhes nas sessões 5.1 e 5.2).

Além da funcionalidade descrita acima, este sistema deverá identificar toda vez que algum aplicativo de terceiros (externo ao projeto) for executado e guardar algumas informações sobre sua execução a serem definidas em tempo de desenvolvimento (tais como horário, data, tempo de utilização do mesmo, etc). Uma vez identificado que um aplicativo foi executado pelo usuário, esta informação deverá ser enviada para o CONTEXT MANAGER, que por sua vez é responsável por adicionar informações de contexto (como localização através dos hardwares previstos) e enviar ao CONTEXT SERVER.

Tendo conhecimento sobre as ações e do ambiente atual do usuário, por exemplo, o aplicativo terá condições de alterar o desktop do dispositivo de maneira a deixar evidente o ícone de um aplicativo qualquer, como e-mail por exemplo.

O aplicativo para controle do desktop será um novo aplicativo de controle de *home screen* do Android, semelhante a vários já conhecidos *Launchers* (exemplos e detalhes nas seções 3.4 e 5.3.2).

4.3.1 Arquitetura

A figura 14 apresenta a arquitetura macro da aplicação, disposta através de seus agentes principais e comunicação com o projeto CONTEXT MANAGER.

[1] *Com ações especializadas, é entendido que são informações coletadas através de preferências captadas de maneira implícita do usuário (como processos em operação) e também explícitas (através do registro de ações que são mais comumente tomadas pelo mesmo).*

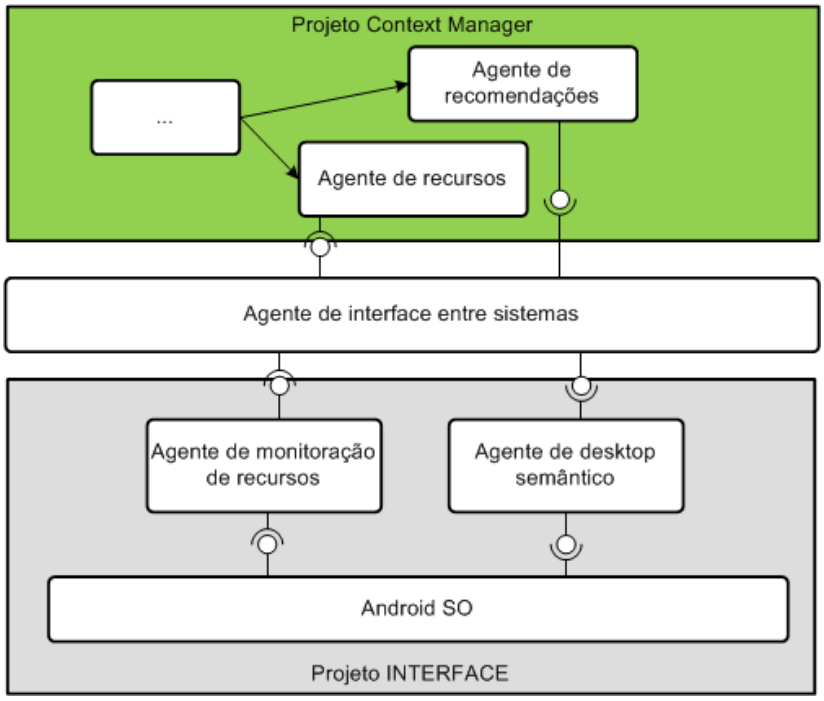


Figura 14 - Arquitetura macro da aplicação e sua comunicação com o projeto Context Manager – criado pelo autor

Através da figura podemos ver que:

- A comunicação entre os dois projetos deverá ser gerenciada por um agente de comunicação, chamado “agente de interface entre sistemas”. Este agente será responsável por transferir os dados entre uma aplicação e outra.
- Os agentes de monitoração de recursos e do desktop semântico terão comunicação direta com o sistema operacional Android.
- O projeto INTERFACE deve enviar dados sobre o comportamento do usuário através do agente de monitoração de recursos para o projeto CONTEXT MANAGER. Isto garantirá que as recomendações poderão ser feitas e, com o tempo, especializadas.
- Para que o agente de desktop semântico opere, ele depende diretamente de uma requisição *push* do projeto CONTEXT MANAGER.

4.3.1.1 Fluxos de dados

Abaixo estão apresentados três cenários de comportamentos da aplicação. Os dois primeiros ilustram atividades isoladas, e o terceiro mostra o cenário completo de uso do ambiente.

- **Cenário 1: Usuário executa um aplicativo qualquer**

A figura 15 apresenta o fluxo de informações executado pela aplicação toda vez em que o usuário executar algum aplicativo qualquer.

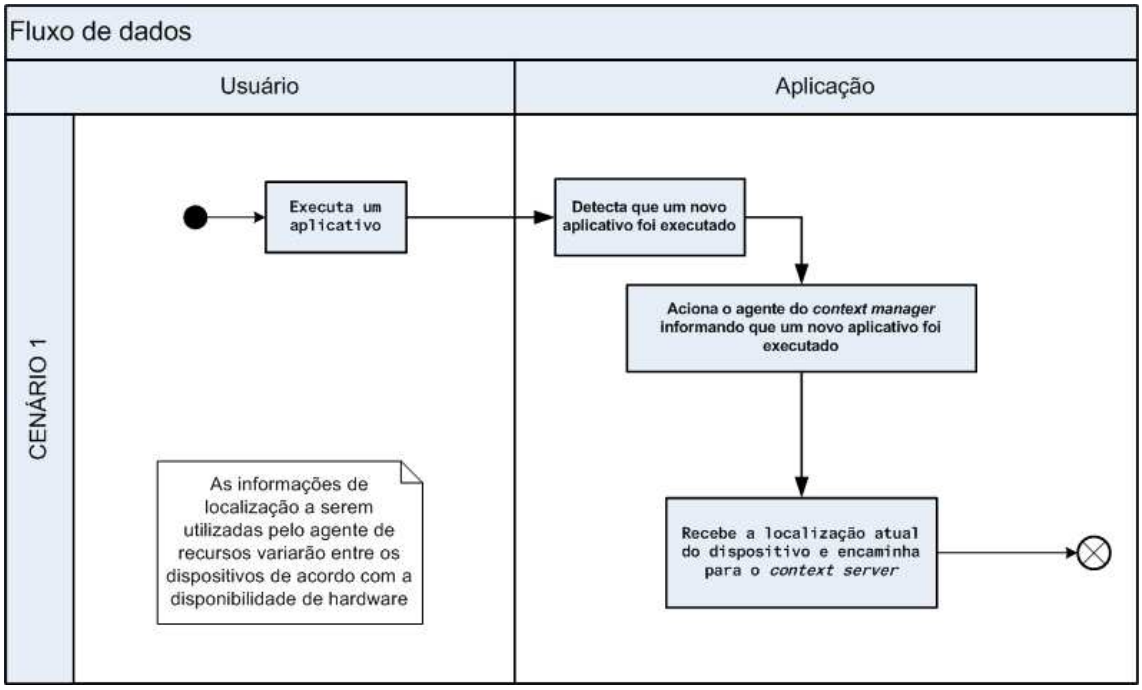


Figura 15 - Fluxo de dados para o cenário 1 – criado pelo autor

A imagem mostra o fluxo que acontecerá em segundo plano no smartphone, e que por isso será imperceptível ao usuário. O lançamento da sua aplicação será feito normalmente e o uso da mesma não será afetado.

Pode também ser visto um comentário que menciona que será feito uso apenas das informações disponíveis em cada modelo de smartphone a depender do hardware disponível. As informações de localização (GPS, IEEE 802.11, bluetooth, acelerêmtro, NFC, etc) e tempo (data e hora) serão utilizadas pelo projeto CONTEXT MANAGER para guardar dados sobre as ações do usuário, e desta maneira especializar as recomendações a serem feitas (descrito no cenário 2). Esta informação também merece destaque, pois a precisão das recomendações poderá variar entre os usuários de diferentes modelos de smartphone que poderão não possuir todos os tipos de sensores e hardwares suportados pelo projeto.

- **Cenário 2: O aplicativo do projeto recebe uma recomendação de adaptação do desktop**

A figura 16 apresenta o fluxo de dados que ocorrerá sempre que o agente desktop semântico receber uma recomendação vinda do agente de recomendação de recursos.

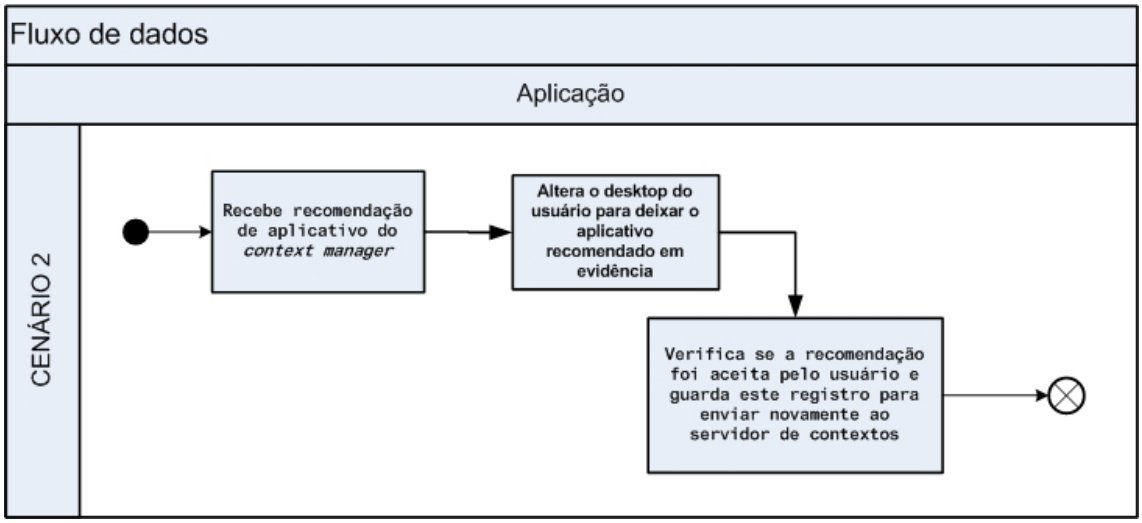


Figura 16 - Fluxo de dados para recepção de recomendação – criado pelo autor

O fluxo mostra o recebimento de uma recomendação. Esta recomendação virá do ambiente do Context Manager e será interpretada a fim de otimizar o desktop para a recomendação recebida (detalhes na seção 6.3.1). O usuário poderá ainda rejeitar uma recomendação, e poderá fazê-lo de duas maneiras diferentes (também descrito na seção 6.3.1).

- **Cenário 3: O aplicativo do projeto recebe uma recomendação de adaptação do desktop**

A figura 17 apresenta o fluxo completo de uso da arquitetura tendo os três projetos integrados. Em uma figura estão descritos dois fluxos de dados entre eles, que mostra a integração entre os mesmos.

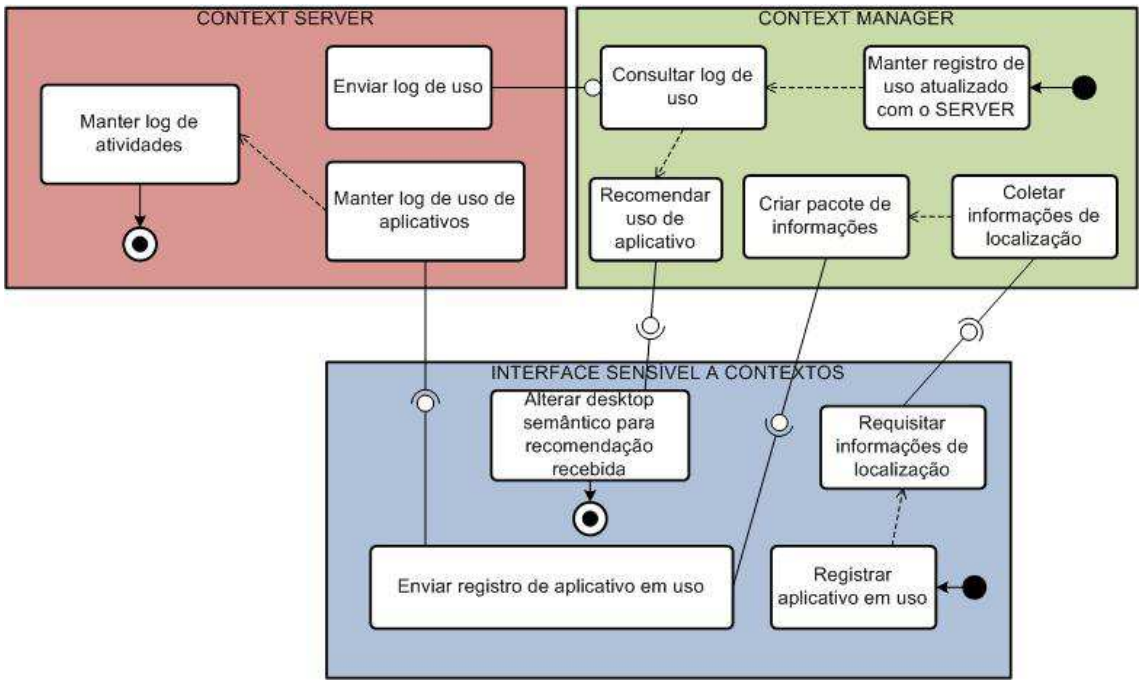


Figura 17 - Fluxo completo de funcionamento da arquitetura – criado pelo autor

A figura 17 mostra os três projetos integrados e as comunicações entre eles. As atribuições dos projetos são as seguintes:

- Inicio 1: a Interface i**dentificará um novo aplicativo em uso** e **requisitará as informações de localização** ao Context Manager. O Context Manager **coletará as informações de localização** e **criará um pacote de informações**, que será devolvido à Interface, que por sua vez, junto de outros dados coletados, o **enviará** ao Context Server, que **manterá o log de uso de aplicativo** e **manterá o log de atividades**.
- Início 2: o Context Server **manterá o registro de uso atualizado**, e para isso **consultará o log de uso existente**. Tendo o log de uso atual, **recomendará o uso de um aplicativo quando o mesmo for oportuno** à Interface. Tendo recebido a recomendação, **o desktop será alterado para deixar a recomendação em evidência**.

4.3.2 O home replacement:

Home replacement é um aplicativo normal em Android que substitui o processo chamado quando usamos o botão "*home*". Há diversos exemplos de aplicativos que substituem o desktop (*home screen*) nativo do Android. Abaixo estão apresentados três exemplos. O primeiro deles é a *home screen* original do sistema (em sua versão 2.2), o segundo é uma customização bastante simples e visa apenas dar melhor performance ao aparelho em que foi instalado. O terceiro parte para uma abordagem mais detalhista e sofisticada, onde prima pela elegância e pela experiência com o usuário.

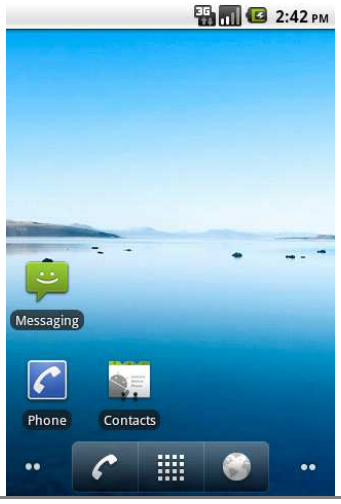
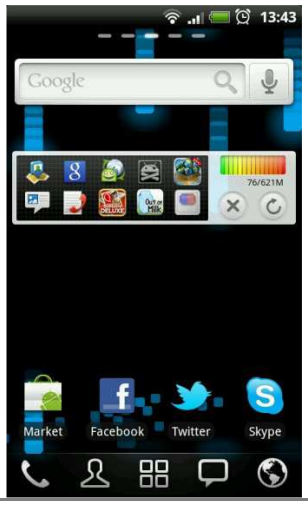
Home screen original	*Home replacement* básico – foco em melhoria de performance para o usuário	*Home replacement* sofisticado – foco em elegância para interação com o usuário
		

Tabela 5 – Exemplos de aplicativos de *home replacement*. Fonte: Android dev center, Go Launcher Ex e ADW Launcher, respectivamente.

Os três exemplos mostrados na tabela 5 usam a mesma técnica a ser empregada neste projeto. Estes aplicativos se diferem por vários aspectos, mas se assemelham no fato de que nenhum deles aborda o ponto que este projeto abordará, que é a automação da alteração do desktop conforme as preferências do usuário. O diferencial neste caso é que, além de assemelhar-se com um desktop comum (como os vistos acima), os ícones e/ou textos sejam dispostos de acordo com a inteligência identificada através das ações especializadas já armazenadas pelo sistema de acordo com os hábitos do usuário.

Em todos os exemplos de *home replacement* apresentados neste projeto há a preocupação com o consumo de bateria, pois o aplicativo de *home* interfere diretamente no consumo da mesma, uma vez que é responsável pela navegação no smartphone (conforme figura 1). Este consumo varia entre cada um deles de acordo com, principalmente, a quantidade de efeitos visuais que cada uma delas usa. Este cuidado com o consumo foi levado em consideração durante o desenvolvimento do projeto as considerações estão aprofundadas na seção 7.4.

O projeto desenvolvido tem, entre outros, o foco de deixar um aplicativo em evidência, de maneira a facilitar sua execução pelo usuário. As maneiras para deixar determinado ícone em destaque foram escolhidas entre as alternativas levantadas abaixo, que foram encontradas de acordo com as limitações do sistema operacional. Algumas maneiras de deixar determinado ícone em destaque seriam:

- Escurecer a tela exibida ao usuário, deixando apenas o ícone desejado em sua cor normal. No exemplo abaixo (figura 18) a tela seria escurecida, deixando apenas o ícone destacado em sua coloração original:

Figura 18 - Exemplo de alteração no desktop – criado pelo autor

- Alterar a fonte do rótulo que acompanha cada ícone na tela inicial. No exemplo abaixo (figura 19), o texto destacado teria sua fonte alterada para dar maior destaque ao ícone. Alterações possíveis seriam alterar a cor do texto, seu tamanho e estilo de texto:

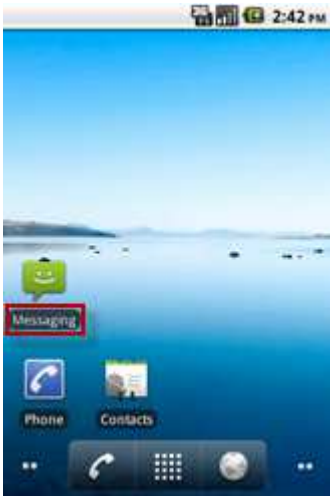


Figura 19 - Exemplo de alteração no desktop – criado pelo autor

- Rearranjar a tela, de maneira a colocar o ícone desejado no centro da mesma, alterando as posições dos ícones restantes por um curto período de tempo (igual ao tempo que a recomendação durar). No exemplo abaixo (figura 20), o ícone de "Messaging" seria posicionado no ponto destacado da figura:

Figura 20 - Exemplo de altearção no desktop – criado pelo autor

Entre as alternativas mostradas, a escolhida foi a terceira delas. Esta escolha foi feita baseada nos seguintes fatores:

- O escurecimento da tela proposto na primeira alternativa poderia causar uma falsa impressão de que algum processamento estaria sendo executado, e com isso o usuário poderia hesitar em executar alguma operação.
- A segunda alternativa, de simplesmente alterar o tamanho do label que descreve o aplicativo recomendado, não se mostra interessante pois não daria destaque suficiente ao mesmo.

- As duas primeiras alternativas têm um problema em comum: o destaque de algum aplicativo que não estivesse presente na tela inicial da *home* não seria percebido pelo usuário a menos que algum outro aviso (sonoro ou não) fosse disparado. Este tipo de aviso não foi usado no projeto pois eles poderiam causar transtornos ao usuário em seu uso normal da *home*.
- A terceira alternativa mostra-se mais atraente entre elas pois não cria os problemas destacados nos três pontos acima. A terceira alternativa poderia ainda causar o problema de esconder os aplicativos que estivessem no local em que o ícone seria destacado. Mas este problema pode ser contornado pelo usuário simplesmente alterando o lugar de onde a recomendação é feita ao arrastar o ícone recomendado.

4.4 Comunicação

O programa (conjunto dos três projetos) encontrou o fator comum de que, entre outros recursos necessários, os três projetos desenvolveram aplicativos Android em seu escopo. Tendo este ponto em vista, as alternativas sobre comunicação dentro do Android e entre diferentes aplicações foram estudadas. Algumas alternativas foram levantadas e a que se mostrou mais útil a este projeto foi escolhida (seção 5.4.4):

4.4.1 Gravação em banco de dados aberto.

- **Descrição:** Consiste em criar registros usando o banco de dados nativo do Android (SQLite) e permitir que os arquivos gerados pelo banco de dados sejam acessados por aplicativos externos. Desta maneira, o aplicativo gerador de dados faria a gravação do registro que precisaria ser lido pelos outros aplicativos interessados.
- **Vantagens:**
    - Alta confiabilidade na armazenagem dos dados.
    - Facilidade na implementação dos cenários de validação, já que desta maneira um registro natural de uso estaria sendo criado (log).
- **Desvantagens:**
    - Altíssimo e desnecessário consumo de bateria por parte do aplicativo receptor da informação. Este receptor precisaria consultar o banco de dados constantemente, o que causaria um consumo excessivo de recursos de hardware do dispositivo (principalmente processador).
    - Necessidade de criação de um diagrama ER, mesmo que minimalista, para que os aplicativos envolvidos soubessem o que consultar. Isto traria uma burocracia também desnecessária ao projeto.
    - Quaisquer alterações no banco precisariam ser revistas em todos os aplicativos.

4.4.2 Gravação em arquivo aberto

- **Descrição:** Consiste em criar arquivos de registros com uma estrutura pré-estabelecida e permitir que os arquivos gerados sejam acessados por aplicativos externos. Desta maneira, o aplicativo gerador de dados faria a gravação do registro que precisaria ser lido pelos outros aplicativos interessados.

- **Vantagens:**

    - Facilidade na implementação.

- **Desvantagens:**

    - Altíssimo e desnecessário consumo de bateria por parte do aplicativo receptor da informação. Este receptor precisaria consultar os arquivos gerados constantemente, o que causaria um consumo excessivo de recursos de hardware do dispositivo.

4.4.3 Chamada de webservice em servidor que notifique as aplicações

- **Descrição:** Consiste em criar um servidor de dados para centralizar a troca de informações entre os aplicativos. Este servidor seria responsável por portar um banco de dados para armazenar dados solicitados e enviados e também disponibilizar webservices para serem acionados pelos aplicativos quando precisassem consultar ou enviar algum tipo de informação.

- **Vantagens:**

    - Confiabilidade e consistência da informação.

- **Desvantagens:**

    - Altíssimo e desnecessário consumo de bateria por parte dos aplicativos receptores das informações. Estes receptores precisariam consultar os webservices constantemente, o que causaria um consumo excessivo de recursos de hardware do dispositivo e também de internet.

    - Grande e desnecessária burocracia sobre o desenvolvimento e sobre o processo.

4.4.4 Android Broadcast – ESCOLHIDO

- **Descrição:** Consiste em criar uma simples classe, usando recursos do sistema operacional, com funcionamento parecido a uma radiodifusão (do inglês *broadcast*). Neste cenário uma classe única (gerador de informação) gera uma transmissão de dados simples (através de um objeto centralizador que contém várias informações) que é enviada a todos os receptores previamente "cadastrados". Os receptores se "cadastram" para receber a informação do *broadcast* gerado e a partir de então receberão alertas através do mesmo conceito de *listeners* (do padrão *observer*) presente em muitas linguagens de programação.

- **Vantagens:**
    - Simplicidade na codificação.
    - Consumo insignificante de bateria do dispositivo envolvido.
    - Simplicidade na documentação.
- **Desvantagens:**
    - Envio de dados limitado.

4.4.5 Android ContentProvider

- **Descrição:** Consiste em criar uma simples classe, usando recursos do sistema operacional, com conceito operacional similar ao broadcast mostrado acima, mas com suporte muito rico com relação a formalização das informações trocadas e possibilidades de transmitir dados de maneiras mais claras. O ContentProvider trabalha de maneira similar a um banco de dados em sua geração e também em seu consumo, e com isso possibilita a criação de métodos para tornar a troca de informações mais robusta como, por exemplo:
    - ***Query:*** *usado para procurar dados entre os dados enviados.*
    - ***Insert:*** usado para inserir dados no pacote a ser enviado.
    - ***Delete:*** usado para excluir dados do pacote de dados.
    - ***Update:*** usado para alterar dados do pacote de dados.
    - ***OnCreate:*** método executado em todas as vezes em que um pacote de dados for inicializado.
- **Vantagens:**
    - Consumo insignificante de bateria do dispositivo envolvido.
- **Desvantagens:**
    - Envio de dados limitado.

4.5 Utilização do hardware estudado

Após o programa ter sido criado e o hardware a ser usado definido e estudado, os participantes escolheram quais deles seriam usados em cada projeto. O projeto Interface usou as informações do hardware de GPS em seu desenvolvimento, conforme mostrado na seção 6.3.2.2, deixando a cargo dos outros projetos o estudo mais profundo sobre a utilização dos restantes.

4.6 Validação

A validação torna-se importante para poder evidenciar a possibilidade de comunicação deste projeto com os outros integrantes do programa, e também para mostrar sua funcionalidade correta. Estão previstos alguns desenvolvimentos auxiliares, além do que foi descrito no item 5.3, para a validação do modelo proposto:

- **Simulação de recomendação:** será desenvolvido um sistema auxiliar para simular a recomendação de uso de um aplicativo para o projeto desenvolvido. Este desenvolvimento é exigido para que possa ser executada uma das principais validações do sistema, que se trata do remanejamento de ícones e evidenciação de um aplicativo.

- **Simulação de uso de aplicativo:** este teste será feito usando o próprio projeto desenvolvido e um auxiliar. O uso de um aplicativo será disparado e os dados gerados pela aplicação sobre o histórico de uso serão coletados. Após a coleta, eles serão enviados ao aplicativo auxiliar, que os receberá e simulará uma localização para retornar à Interface.

- **Identificação da receptividade do usuário:** após uma recomendação ser disparada na tela do dispositivo, o programa deverá monitorar o comportamento do usuário. Caso o mesmo altere algo relacionado à recomendação, esta ação deve ser informada ao recomendador, sendo que alguns cenários poderão ser identificados:

    - Caso o usuário mova a recomendação para um lugar ainda em destaque da tela, a aplicação será calibrada para que coloque as recomendações no local que o usuário escolheu e não mais no local padrão.

    - Caso o usuário mova a recomendação para algum local de menos visibilidade no desktop, deverá ser enviado um aviso para o recomendador, indicando que a recomendação não foi bem recebida. Com isto espera-se que aquela recomendação seja suspensa até que o sistema se torne mais especializado.

    - Caso o usuário exclua o ícone da aplicação, a mesma notificação será enviada ao recomendador para que aquela recomendação não seja repetida.

- **Interferência na bateria:** a economia de bateria causada automaticamente pelo uso do aplicativo desenvolvido será mensurada e seus benefícios e eventuais problemas serão evidenciados.

5. Desenvolvimento

Esta seção apresenta como o desenvolvimento do projeto foi feito e também como as ferramentas, técnicas, e outros artefatos apresentados foram usados.

O projeto foi desenvolvido usando um código da Google disponível na web para facilitar a customização e evitar retrabalho, que está apontado na seção 6.1.

Algumas técnicas de programação foram empregadas no projeto para possibilitar e também simplificar a execução de algumas tarefas necessárias ao funcionamento, que podem ser vistos na seção 6.2.1.

5.1 Código base para o projeto

Para o desenvolvimento do projeto, um *launcher* customizado a partir da versão inicial da Google foi usado. Este *launcher*, que serviu como base, possui seu código sob a licença *Apache*, que pode ser lida através da URL 1[2].

O código usado consiste de uma customização feita por outros desenvolvedores sobre a versão 2.1 (codinome Eclair – ver figura 22) da home original presente no Android. O código pode ser encontrado por qualquer pessoa interessada através da URL 2[3]:

Com o desenvolvimento tendo sido feito sobre a versão 2.1 do Android, é garantido, pelo site da mantenedora do Android, que, atualmente, 99,8% dos dispositivos operando possibilitam a instalação da ferramenta criada, conforme fonte mostrada na figura 23, atualizada na data de 17 de Janeiro de 2013:

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.4%
2.2	Froyo	8	9.0%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	47.4%
3.1	Honeycomb	12	0.4%
3.2		13	1.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.1%
4.1	Jelly Bean	16	9.0%
4.2		17	1.2%

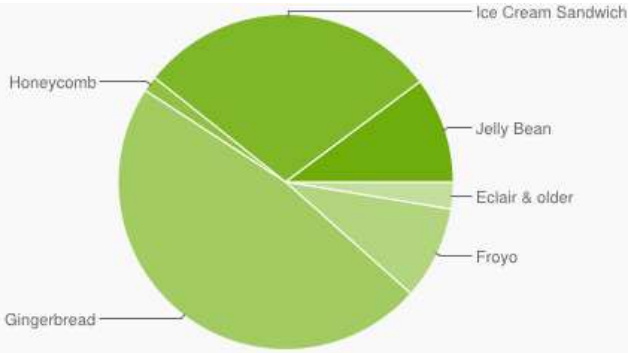


Figura 21 - Distribuição de versões do Android sobre dispositivos operantes. Fonte: Android dev center.

[2] URL 1 - http://www.apache.org/licenses/LICENSE-2.0
[3] URL 2 - http://code.google.com/p/android-launcher-plus/

5.2 Estrutura do projeto

Tendo o código base em mãos, os requisitos foram levantados de maneira a ordenar o desenvolvimento e então poder pensar nas técnicas que seriam empregadas no mesmo. Foram levantados dois requisitos macro para o projeto, que são:

- Recebimento de recomendação e alteração do desktop.
- Envio de registro de uso junto do pacote de informações quando novo aplicativo for executado.

Os arquivos desenvolvidos durante o projeto foram organizados entre os já existentes do código obtido e dispostos nos *packages* "homeAdds" e "tasks" ficam evidentes através da figura 24:

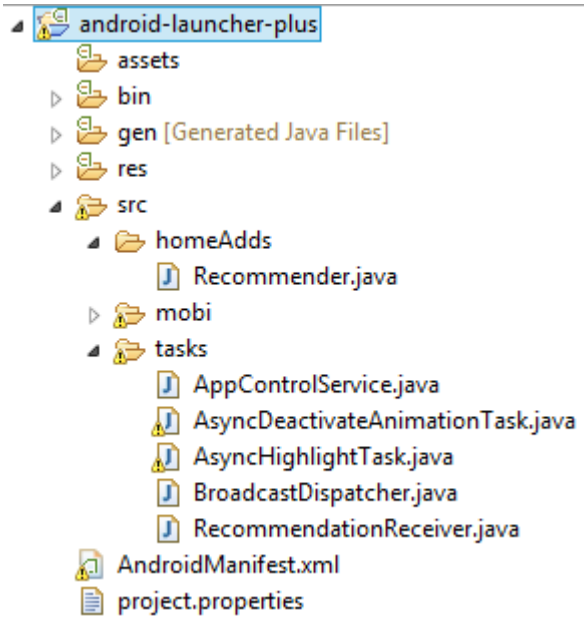


Figura 22 - Árvore de arquivos gerados para o projeto. Fonte: criado pelo autor

A figura 24 apresenta:

- As pastas "res" e "mobi" são os arquivos reaproveitados do projeto original, que possuem classes necessárias para criar o desktop normal ao qual o usuário está acostumado.
- As pastas "assets", "bin" e "gen" são pastas de arquivos gerados automaticamente pelo ambiente de desenvolvimento e são relativas aos recursos que o usuário usou no seu aplicativo (como imagens, áudios, etc).
- As pastas "homeAdds" e "tasks" foram criadas por este projeto e possuem as classes necessárias para tornar as funcionalidades possíveis como foram propostas.

5.2.1 Técnicas usadas:

Para o desenvolvimento do projeto, foram usadas algumas técnicas de programação para possibilitar sua execução. São padrões de desenvolvimento bastante conhecidos e passíveis de serem utilizados em muitas linguagens de programação existentes.

Em especial entre eles, a opção por utilizar o padrão *observer* (*broadcast*) foi importante pois possibilita muitos desenvolvimentos futuros. Desta maneira, como explicado abaixo, a informação gerada por este projeto é enviada a N outros aplicativos, bastando apenas que eles estejam previamente "cadastrados" para receber as informações (descrito na seção 6.3.2).

Abaixo estão listadas as técnicas seguidas de uma breve explicação sobre cada uma delas:

- **Singleton:** padrão de desenvolvimento que limita a classe implementadora a ter apenas uma instância de si própria criada a cada execução da aplicação (Foy, 2002). É recomendado usar singleton sob algumas circunstâncias:

    a) Variáveis globais enviam dados entre diferentes partes da aplicação – cenário usado pela aplicação e explicado na seção 6.3.1.

    b) Um recurso só pode ser usado uma vez – cenário não usado pela aplicação.

    c) Muitas instâncias usadas em diferentes partes da aplicação possuem o mesmo conteúdo ou representam a mesma coisa – este cenário seria criado na aplicação caso *singleton* não fosse aplicado.

- **Broadcast / Observer:** padrão de desenvolvimento que usa dois atores: "dados" e "recebedor". É uma relação de "um para muitos", onde um mesmo disparo atende a "n" recebedores (observers). Observers podem ser usados sob algumas circunstâncias (Xu, 2009):

    a) Quando a abstração da aplicação possui dois objetivos independentes. A separação entre estes objetivos aumentará a chance de encapsulamento – um dos motivos pelos quais é importante neste projeto é que uma aplicação executa um trecho de código, e a outra aplicação está interessada em executar outra parte restante.

    b) Quando um autor de dados não sabe exatamente para quantos receptores deve enviá-los – também presente nos motivos por ter sido implementado neste projeto, onde não se sabe exatamente quantos aplicativos têm interesse em receber as informações geradas pelo mesmo.

    c) Quando o autor dos dados precisa estar habilitado a notificar seus receptores de informações mesmo quando não sabe exatamente quem eles são – também presente neste projeto, pois é feita uma abstração à formalidade dos dados enviados aos outros aplicativos e não se sabe exatamente como cada um deles está esperando pela informação.

- **Background tasks:** através de *tasks* (em português tarefas) em execução em background é possível executar tarefas ainda importantes ao sistema e permitir que a atenção principal do programa seja preservada. Tarefas em segundo plano são importantes em alguns cenários:

    a) Manter o programa responsivo mesmo quando é necessária a execução de uma tarefa que consuma muito tempo de processamento e que poderia, em potencial, fazer a aplicação ficar instável – cenário não usado neste projeto.

b) Executar uma tarefa que precisaria interromper a execução dos comandos do usuário para executar alguma ação que ele não está diretamente interessado – cenário usado neste projeto para permitir que o efeito de aparecimento da recomendação seja possível e não faça com que o aplicativo como um todo pare de responder aos comandos do usuário por um breve período de tempo.

- **Beans:** beans são usados essencialmente para guardar informações e torná-las acessíveis de modo simples. Beans são usados para (Offutt, 2010):

    a) Alterar e obter informações – usado neste projeto para guardar, em memória, qual recomendação foi recebida ou ainda qual aplicativo foi lançado pelo usuário.

    b) Executar eventos entre diferentes objetos – não usado neste projeto.

    c) Criar instâncias de objetos – usado neste projeto para criar objetos relacionados às recomendações.

    d) Guardar objetos usando serialização – também não usado neste projeto.

5.3 Artefatos desenvolvidos

Esta seção mostrará como foram desenvolvidos os principais passos para realizar as principais atividades do ambiente construído neste projeto e algumas das classes principais envolvidas no processo estão relacionadas abaixo com seu funcionamento detalhado. A seção foi dividida em dois pontos identificados como os principais requisitos do sistema:

5.3.1 Recebimento de recomendações e alteração no desktop

Esta seção apresenta o cenário descrito no fluxo presente na figura 15.

Através do armazenamento de dados que são gerados na seção 6.3.2, o projeto context manager, possui autonomia para gerar recomendações a qualquer momento que considerar oportuno. Estas recomendações são interpretadas e então é disparada uma recomendação (reorganização no desktop) para que o usuário veja.

O recebimento de recomendações é feito através de um *observer* usando broadcast. No primeiro método chamado dentro da aplicação, ou seja, logo que a aplicação é iniciada, este *observer* é cadastrado usando um comando nativo do Android para receber as recomendações externas (este processo também é usado pelo item 6.3.2.2):

```
registerReceiver(new AppControlService(), new
IntentFilter(BroadcastDispatcher.recommendationReceiver));
```

O comando consiste em cadastrar uma nova instância da classe “AppControlService” como recebedora de qualquer broadcast que seja enviado no sistema com identificação igual à string “recommendationReceiver” da classe “BroadcastDispatcher”. Então, a cada momento em que o

aplicativo context manager disparar uma nova recomendação, ela será capturada pela classe "RecommendationReceiver" mostrada abaixo na figura 25:

```
1  package tasks;
2
3  import homeAdds.Recommender;
8
9  public class RecommendationReceiver extends BroadcastReceiver {
10
11     public void onReceive(Context context, Intent intent) {
12         if(intent.getAction().equals(BroadcastDispatcher.recommendationReceiver)){
13             // 1
14             String appTitle = intent.getStringExtra(BroadcastDispatcher.appTitle);
15
16             // 2
17             ApplicationInfo info = new ApplicationInfo();
18             info.setTitle(appTitle);
19             Recommender rec = Recommender.getInstance();
20             rec.setRecommendation(info);
21         }
22     }
23 }
```

Figura 23 - Classe responsável por receber as recomendações. Fonte: criado pelo autor

O método *onReceive* verificará se o comando recebido é mesmo uma recomendação recebida, válida e aguardada, e então executará os processos abaixo, seguindo os números presentes na figura:

1) Recebe o nome do aplicativo que foi recomendado.

2) Cria um novo objeto que identifica uma informação de aplicativo e dispara a mesma usando a linha 20. A linha 19 possui o conceito de *singleton* apresentado na seção 6.2.1.

As recomendações são feitas usando tarefas em background para gerar um efeito de destaque no ícone do aplicativo que está sendo recomendado. Este método para destacar o aplicativo foi escolhido entre os sugeridos na seção 5.3.2. Este efeito faz com que o ícone do aplicativo recomendado fique piscando para que o usuário possa vê-lo (exemplo nas imagens 24, 25 e 26). O aplicativo permanece piscando por 1 minuto e após isto continua no centro da tela. Ele sairá do centro da tela após 5 minutos passados da recomendação original. Abaixo está apresentado um exemplo de seu 1funcionamento:

1	2	3
		
Figura 24 - Recomendação - 1	Figura 25 - Recomendação - 2	Figura 26 - Recomendação 3

Após a criação de uma recomendação, caso o usuário entenda que ela foi inoportuna, ele possui duas formas de recusá-la. Esta ação é identificada e enviada ao projeto Context Manager afim de especializar as recomendações. As alternativas para recusa da recomendação são:

- Arrastar o ícone para alguma das telas secundárias do desktop.
- Excluir o ícone (da mesma maneira como faria para excluir um atalho).

5.3.2 Envio de registro de uso junto do pacote de informações quando novo aplicativo for executado

Esta seção apresenta o cenário de quando um novo aplicativo é executado, conforme figura 14 no item 5.3.1.1 e foi divida em duas partes:

- **Envio de registro de execução:** o usuário executa um novo aplicativo qualquer, a aplicação detecta esta execução e informa o projeto *context mananger*.
- **Recebimento da localização e envio do registro completo:** a aplicação recebe um *broadcast* do projeto *context manager* com dados de localização (descritos nas seções 2.5 e 2.6). Estas informações são acrescidas do nome do aplicativo e então enviadas ao projeto *context server*.

5.3.2.1 Envio de registro de execução

Sempre que um novo aplicativo for executado pelo usuário, esteja ele na tela principal do dispositivo ou ainda em um menu secundário, é gerado um alerta para que o aplicativo *context manager* o receba e devolva os dados de localização, que é feito pela classe mostrada abaixo. Este processo é feito pela classe “BroadcastDispatcher” e detalhado na figura 27 abaixo:

```
1  package tasks;
2
3  import mobi.intuitit.android.p.launcher.ApplicationInfo;
6
7  public class BroadcastDispatcher {
8
9      // 1
10     public static String appTitle = "App.Title";
11     public static String appPackageName = "App.PackageName";
12     public static String appExecutionCount = "App.ExecutionCount";
13
14     public static String localizationReceiver = "BROADCAST_LOCALIZACAO";
15     public static String dataPackageSender = "com.unisinos.AppDispatcher";
16     public static String recommendationReceiver = "BROADCAST_RECOMENDACAO_RECURSO";
17
18     public static String POSICAO_LATITUDE = "POSICAO_LATITUDE";
19     public static String POSICAO_LONGITUDE = "POSICAO_LONGITUDE";
20     public static String POSICAO_ALTITUDE = "POSICAO_ALTITUDE";
21
22     // 2
23     public static void dispatchBroadCast(ApplicationInfo app){
24         Intent alert = new Intent(BroadcastDispatcher.dataPackageSender);
25         alert.putExtra(BroadcastDispatcher.appTitle, app.getTitle());
26         alert.putExtra(BroadcastDispatcher.appPackageName, app.getPackageName());
27         alert.putExtra(BroadcastDispatcher.appExecutionCount, app.getLaunchTimes());
28         Launcher.activity.sendBroadcast(alert);
29     }
30 }
```

Figura 27 - Classe para criação de registro de uso – Fonte: criado pelo autor

Os componentes desta classe são divididos em:

1) **Constantes:** variáveis definidas como globais e públicas que podem ser usadas por todas as classes do projeto. Representam constantes únicas e compartilhadas entre os três projetos que são usadas como identificadores para o envio e recebimento de informações entre os *broadcasts*.

2) **Método *dispatchBroadcast*:** é o método responsável por enviar as informações sobre qual aplicativo foi executado naquele momento. As linhas 24 a 27 são responsáveis por enviar os dados de a) nome do aplicativo – usado para identificação do mesmo, b) nome do pacote – usado como chave única, já que vários aplicativos podem possuir um mesmo nome e c) número de vezes que o aplicativo foi executado no dia. A linha 28 executa efetivamente o envio do *broadcast*, que será capturado pelo *context manager* e terá seguimento no processo conforme mostrado abaixo (seção 6.3.2.2).

5.3.2.2 Recebimento da localização e envio do registro completo

Assim como mostrado na seção 6.3.1, um novo *observer* precisa ser registrado para receber a localização do dispositivo como resposta. O registro é feito através do comando abaixo e também é feito no primeiro método executado pelo aplicativo logo que é iniciado:

```
registerReceiver(new AppControlService(), new
IntentFilter(BroadcastDispatcher.localizationReceiver));
```

Quando a notificação é enviada (como descrito pela seção acima, 6.3.2.2), imediatamente o aplicativo context manager recebe os dados e consulta informações sobre localização. Com os dados sobre localização em mãos, o context manager faz um novo broadcast que é recebido pela interface, e está descrito abaixo.

Quando a nova informação é recebida, a classe "AppControlService" interfere e executa alguns passos expostos e detalhados abaixo na figura 28.

```
1  package tasks;
2
3  import mobi.intuitit.android.p.launcher.Launcher;
7
8  public class AppControlService extends BroadcastReceiver{
9
10     @Override
11     public void onReceive(Context context, Intent intent) {
12         if(intent.getAction().equals(BroadcastDispatcher.localizationReceiver)){
13             // 1
14             String appTitle = intent.getStringExtra(BroadcastDispatcher.appTitle);
15
16             // 2
17             String latitude = intent.getStringExtra(BroadcastDispatcher.POSICAO_LATITUDE);
18             String longitude = intent.getStringExtra(BroadcastDispatcher.POSICAO_LONGITUDE);
19             String altitude = intent.getStringExtra(BroadcastDispatcher.POSICAO_ALTITUDE);
20
21             // 3
22             Intent alert = new Intent(BroadcastDispatcher.dataPackageSender);
23             alert.putExtra(BroadcastDispatcher.appTitle, appTitle);
24             alert.putExtra(BroadcastDispatcher.POSICAO_LATITUDE, latitude);
25             alert.putExtra(BroadcastDispatcher.POSICAO_LONGITUDE, longitude);
26             alert.putExtra(BroadcastDispatcher.POSICAO_ALTITUDE, altitude);
27             Launcher.activity.sendBroadcast(alert);
28         }
29     }
30 }
```

Figura 28 - Código para recebimento de dados de localização – Fonte: criado pelo autor

O método "onReceive" verifica se a informação recebida é aquela que está sendo esperada pelo programa comparando o nome do *broadcast* recebido com aquele que está sendo esperado. Os itens apresentados com números estão explicados abaixo:

1) Entre as informações recebidas, a linha 14 obtém o título do aplicativo enviado.

2) As linhas 17, 18 e 19 obtêm as informações de localização enviadas pelo *context manager*.

3) As linhas 22 a 26 montam um novo pacote de informações com os dados enviados pelo *context manager* junto do nome do aplicativo que foi executado anteriormente.

4) A linha 27 executa o envio do broadcast para que o aplicativo *context server* os receba.

6. Validação

Esta seção apresenta os experimentos que foram feitos com o aplicativo a fim de validar seu funcionamento e todas as características desenvolvidas. Para tanto, um novo aplicativo, denominado "testador" foi desenvolvido para simular o envio e recebimento de comandos que os outros sistemas do programa fariam. Cada um destes passos foi mantido em logs que estão mostrados em seus respectivos detalhamentos abaixo.

6.1 Simulação de recomendação

As recomendações foram simuladas pelo testador e recebidas pela interface. Os registros foram obtidos através dos logs gerados pela aplicação no console do aplicativo usado para desenvolvimento.

Abaixo estão relacionadas três diferentes recomendações, que ocasionam um mesmo cenário de testes, feitas em sequência pelo aplicativo testador, e então o resultado obtido na interface:

- Teste 1:
  - Figura 29 – Recomendação feita (texto: Recomendação para Gmail enviada):

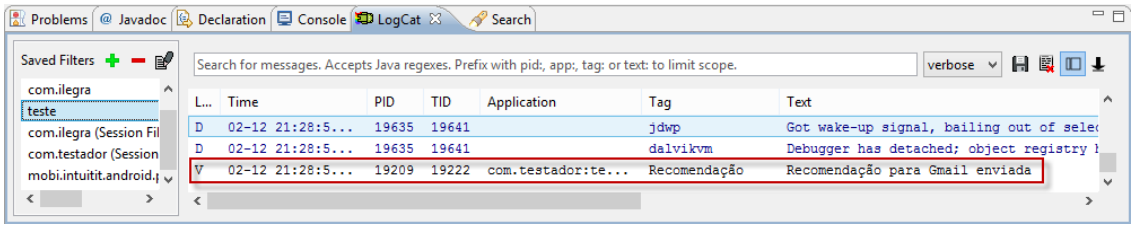


Figura 29 - Recomendação executada. Fonte: criado pelo autor

  - Antes e depois (figuras 30 e 31):

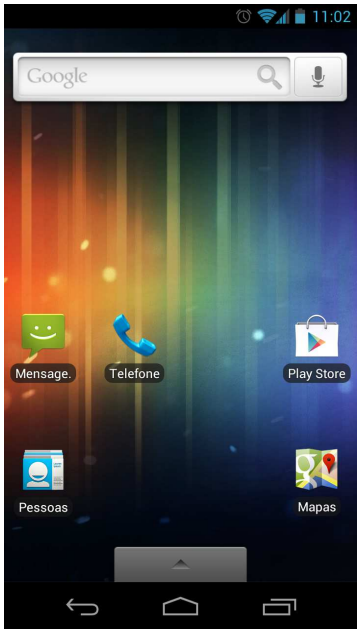


Figura 30 - Home sem recomendação. Fonte: criado pelo autor

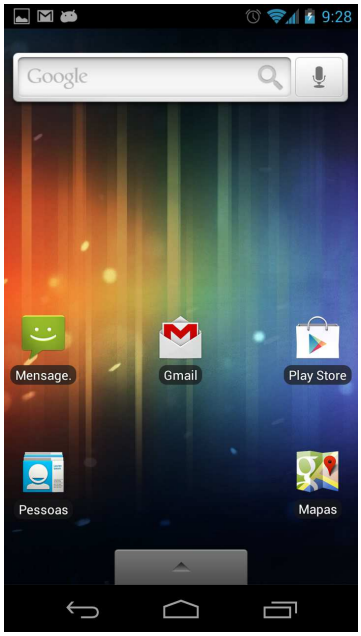


Figura 31 - Recomendação do aplicativo Gmail na interface. Fonte: criado pelo autor

- Teste 2:
  - Figura 32 – Recomendação feita (texto: Recomendação para Mapas enviada):

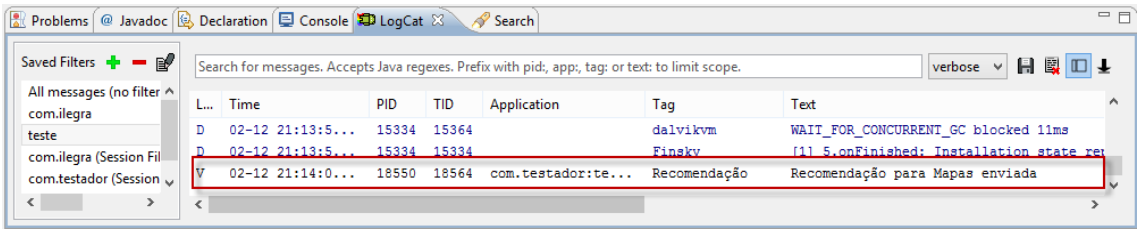


Figura 32 - Recomendação executada. Fonte: criado pelo autor

  - Antes e depois (figuras 33 e 34):

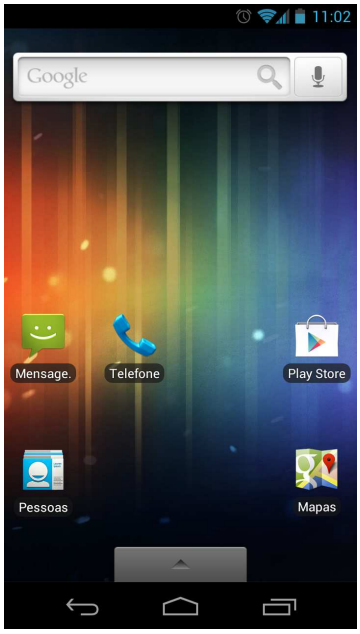


Figura 33 - Home sem recomendação. Fonte: criado pelo autor

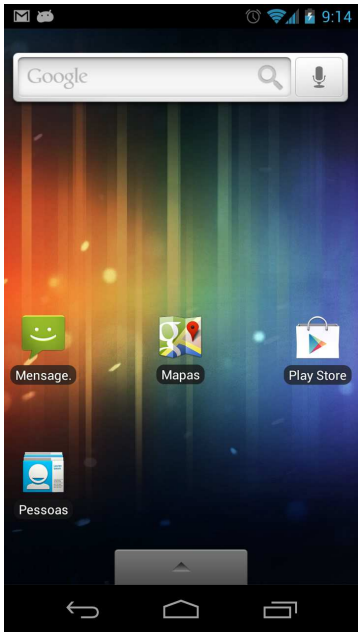


Figura 34 - Recomendação do aplicativo Mapas na interface. Fonte: criado pelo autor

- Teste 3:
  - Figura 35 – Recomendação feita (texto: Recomendação para Angry Birds enviada):

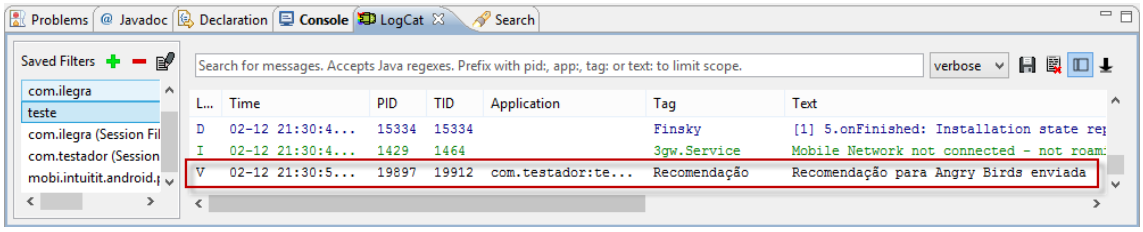


Figura 35 - Recomendação executada. Fonte: criado pelo autor

  - Antes e depois (figuras 36 e 37):

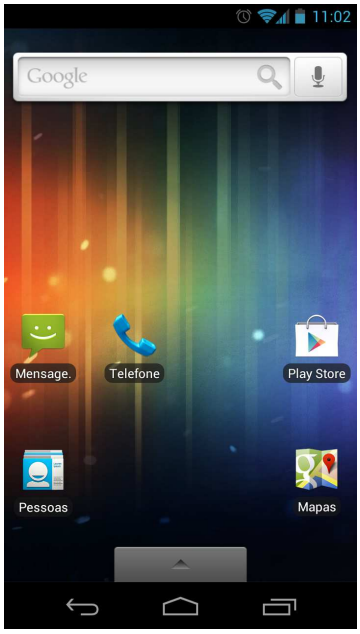


Figura 36 - Home sem recomendação. Fonte: criado pelo autor

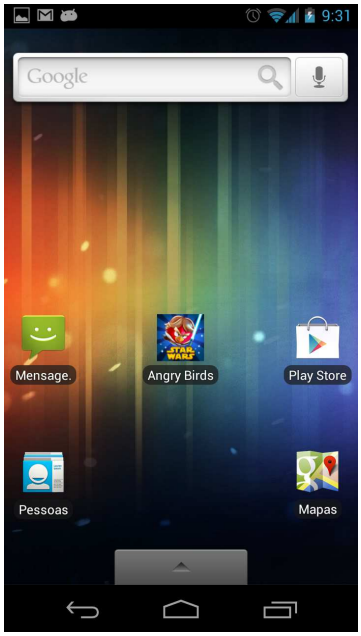


Figura 37 - Recomendação do aplicativo Angry Birds na interface. Fonte: criado pelo autor

A recomendação é mantida piscante por 1 minuto e ativa por 5 minutos. Após passado o primeiro minuto, o efeito que faz com que o ícone permaneça piscando é removido, mas o ícone continua em destaque na tela inicial. Tendo passado os 4 minutos restantes, a interface volta a ter a organização dos ícones como era antes da recomendação.

6.2 Simulação de uso de aplicativo

A simulação de usos de aplicativos foi feita partindo do aplicativo de interface e sendo recebida pelo testador. Também foram executados três diferentes testes e o recebimento das informações mostrado novamente através dos logs da aplicação:

- Teste 1:
    - Execução de um aplicativo existente na tela inicial do usuário (onde geralmente ficam os aplicativos favoritos e mais utilizados) – figura 38.

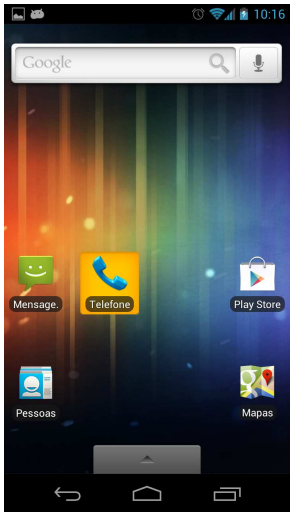


Figura 38 - Aplicativo Telefone executado. Fonte: criado pelo autor

- Recepção pelo testador e envio das informações sobre localização executados – figura 39. Texto: 1 – Aplicativo Telefone lançado e recebido. 2 – Localização atual enviada.

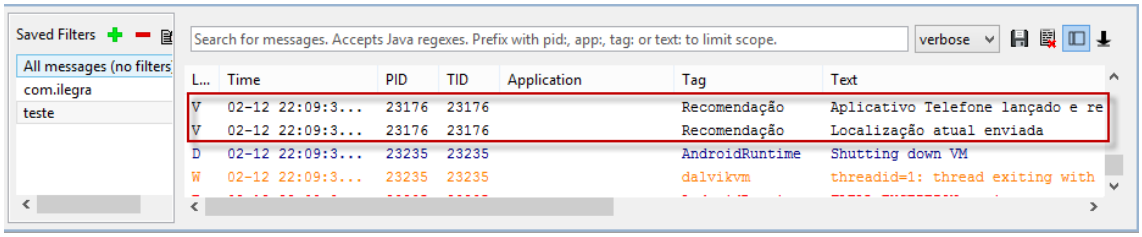


Figura 39 - Execução de aplicativo recebida e respondida. Fonte: criado pelo autor

- Teste 2:
  - Execução de um aplicativo no menu do que contém todos os aplicativos instalados – figura 40.

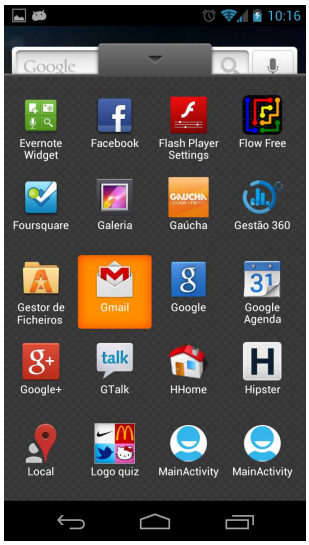


Figura 40 – Aplicativo Gmail executado. Fonte: criado pelo autor

- Recepção pelo testador e envio das informações sobre localização executados – figura 41. Texto: 1 – Aplicativo Gmail lançado e recebido. 2 – Localização atual enviada.

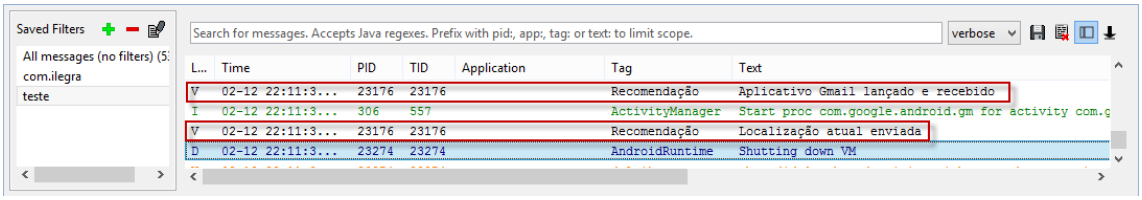


Figura 41 - Execução de aplicativo recebida e respondida. Fonte: criado pelo autor

- Teste 3:

- Execução de um aplicativo no menu do que contém todos os aplicativos instalados – figura 42.

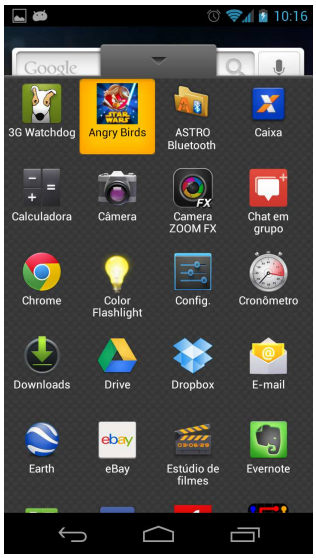


Figura 42 – Aplicativo Angry Birds executado. Fonte: criado pelo autor

- Recepção pelo testador e envio das informações sobre localização executados – figura 43. Texto: 1 – Aplicativo Angry Birds lançado e recebido. 2 – Localização atual enviada.

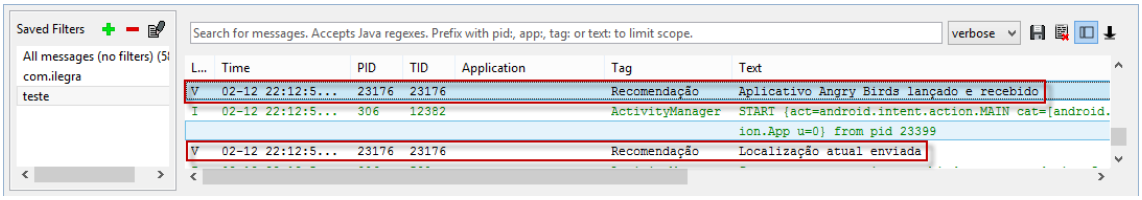


Figura 43 - Execução de aplicativo recebida e respondida. Fonte: criado pelo autor

6.3 Identificação da receptividade da recomendação

Tendo visto na seção 7.2 como os dados de histórico são gerados e também como eles são recomendados na seção 7.1, a preocupação sobre a interferência nas atividades normais do usuário surgiu. Então se tornou importante verificar se o uso da ferramenta é bem recebido pelo usuário para então poder especializar cada vez mais as recomendações que são enviadas a ele.

Pensando nisso, após uma recomendação qualquer ter sido feita, o aplicativo de interface ficará atento a caso o usuário altere de alguma maneira a recomendação que lhe foi feita. Para isto, testes sobre a movimentação do ícone de recomendação foram feitos, e a cada vez em que este evento foi identificado, um novo sinal ao sistema testador foi enviando indicando que a recomendação não foi bem recebida.

Para o cenário criado no teste #3 da seção 7.1 descrito acima, o ícone recomendado (do jogo *Angry Birds*) foi movido para uma das outras telas existentes no aplicativo interface, mas que não é a mesma que a recomendação gerou. Isto criou um evento que foi enviado ao testador e é mostrado abaixo através da figura 44:

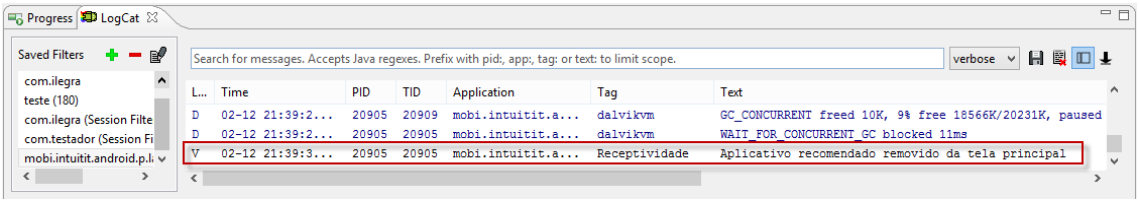


Figura 44 - Recomendação recebida e rejeitada pelo usuário. Texto: Aplicativo recomendado removido da tela principal. Fonte: criado pelo autor

Através deste cenário, onde o usuário poderá rejeitar uma recomendação gerada automaticamente pelo programa, espera-se dar insumos para que o recomendador adquira maior especialidade e então possa fazer recomendações mais extas.

6.4 Interferência na bateria

Apesar de não ter sido o foco inicial deste trabalho, foi identificado que o consumo de bateria poderia ter um ganho com o uso do mesmo. O consumo de bateria foi destacado pois espera-se obter uma economia, mesmo que não muito expressiva, através das recomendações.

Com o tempo de uso do ambiente como um todo, o agente recomendador naturalmente se especializará com relação às recomendações feitas a cada usuário. Com isto, as tarefas rotineiras e mais repetitivas serão recomendadas com precisão e farão com que o usuário não gaste o tempo que geralmente gastaria para encontrar o ícone do seu aplicativo. Este ganho será obtido com a economia de tempo para identificar onde um determinado aplicativo está localizado na interface, tarefa esta que não será mais necessária, pois o mesmo já estará disponível e em destaque assim que o usuário ativar a tela do dispositivo.

A economia de bateria será feita através das recomendações, como exemplificado na tabela 7 abaixo. Com o ícone em destaque, o tempo que o usuário gastará para encontrar o aplicativo que deseja executar será reduzido, e a redução do tempo de procura do aplicativo, o tempo de atividade

(em luminescência) da tela será reduzido (conforme mostrado na seção 2.2.), e então a bateria será menos exigida.

1	2	3
		

Tabela 6 – Exemplo de aplicativo recomendado. Fonte: criado pelo autor

Esta economia dependerá diretamente da habilidade do usuário com seu dispositivo para poder representar algum ganho. Este ganho de bateria será maior, por exemplo, para usuários com mais dificuldades em assimilar as diferentes interfaces dos dispositivos existentes. Aqueles usuários que não costumam usar atalhos em seus dispositivos também teriam um grande ganho com este ambiente, pelo fato de que o mesmo se encarregará de cria-los e excluí-los automaticamente.

7. Conclusão

O MyFace apresenta um diferencial entre todas as arquiteturas existentes, que se trata das recomendações geradas para facilitar o uso do smartphone. Com este uso, também se obteve um ganho de bateria, tratando um dos grandes problemas dos dispositivos móveis atuais.

O trabalho apresentado soluciona o problema levantado pela questão de pesquisa, "**como se pode tornar mais rápido o uso de um smartphone através da interferências em seu desktop?**"**,** por mostrar uma arquitetura capaz de tornar mais rápido o uso de smartphones por usuários comuns através da recomendação espontânea de aplicativos.

Com o trabalho desenvolvido apresentado, foi visto que uma robusta arquitetura foi criada para recomendação de usos de aplicativos em desktops de dispositivos móveis. A arquitetura mostra-se robusta por abranger mais de 99% dos usuários da plataforma Android, o que a garante um fácil efeito de disseminação, e também por usar parte do código oficial da Google, mantenedora do Android, o que garantirá que o aplicativo não terá falhas em seu uso.

As recomendações, disparadas através de estudo sobre o histórico de uso registrado, são mostradas de maneira clara ao usuário, que pode optar por aceita-las ou por rejeitá-las, o que dispara um processo para que a mesma recomendação não seja repetida.

A ideia de desenvolvimento desta plataforma também mostra-se possível para outros sistemas operacionais móveis, desde que com suas respectivas alterações necessárias, todas elas relacionadas à viabilidade técnica restringida pela mantenedora da plataforma.

Em adição à tabela comparativa mostrada na seção 3.5, o fato do MyFace gerar recomendações para seus usuários e apresentar economia no uso da bateria criam seu grande diferencial entre todas as bibliografias encontradas. Após o desenvolvimento, a tabela foi revista e o MyFace considerado na comparação, que pode ser vista através da nova tabela 7 abaixo, com seus critérios de comparação descritos a seguir:

1Interfere no desktop – indica se o trabalho relacionado executa qualquer ação que interfere de alguma maneira no desktop que o usuário já está acostumado a usar.

2Substitui o desktop – indica se o trabalho relacionado substitui o desktop do usuário como um todo, e não apenas altera alguns pontos para fins de visualização.

3Organiza o desktop – indica se o trabalho relacionado altera a forma como os dados com os quais o usuário interage normalmente são apresentados.

4Otimiza o desktop – indica se o trabalho relacionado acelera ou facilita de alguma maneira processos que o usuário esteja acostumado a realizar.

5Recomenda usos no desktop – indica se o trabalho relacionado indica ou sugere quaisquer tipos de processos no desktop do usuário.

6Apresenta economia de bateria – indica se o trabalho relacionado apresenta alguma economia no consumo de bateria.

	1	2	3	4	5	6
Parctab		X				
Widgets	X	X		X		
Ambiente residencial			X			
Framework para Windows Phone	X		X			
Buscas Móveis	X					
Framework genérico				X		
Otimização de carregamento	X	X		X		
Fences	X		X			
LauncherPro	X	X	X	X		
ADWLauncher	X	X	X			
MyFace	X	X	X	X	X	X

Tabela 7 – Nova comparação entre trabalhos relacionados

8. Trabalhos futuros

Alguns passos foram pensados durante o desenvolvimento deste trabalho e foram colocados na seção de trabalhos futuros devido a sua complexidade ou tempo insuficiente para desenvolvimento durante esta primeira iniciativa. São algumas melhorias no ambiente para que fique mais intuitivo e também para trazer mais funcionalidades ao mesmo. São elas:

- Criar configurações para que o usuário possa definir por quanto tempo uma recomendação é mostrada.
- Criar funcionalidade para “silenciar” (suspender) recomendações temporariamente.
- Especificar como a recomendação foi rejeitada e atribuir pesos à rejeição, com intuito de especializar as recomendações do Context Manager e enviá-lo estes dados.
- Criar mais funcionalidades no aplicativo, a fim de deixa-lo mais amigável ao usuário em comparação aos já encontrados na bibliografia.

9.

10. Bibliografia

Adams, N., Schilit, B. N., Gold, R., & Tso, M. M. (n.d.). An Infrared Network for Mobile Computers. *Control*.

Alves, S. (2006). A matemática do GPS, *1*.

Chu, D., Kansal, A., Liu, J., & Zhao, F. (2012). Mobile Apps: It ' s Time to Move Up to CondOS, 1–5.

Daquino, F. (2012). 15 programas para personalizar o Android. Retrieved from http://www.tecmundo.com.br/pdf/18637-15-programas-para-personalizar-o-android-video-.pdf

Dey, A. K., Abowd, G. D., & Salber, D. (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, *16*(2), 97–166. doi:10.1207/S15327051HCI16234_02

ECMA. (2005). Near Field Communication White paper, *012*.

Falcon, A. (2012). Top 10 desktop customization tools for cleaner Windows. Retrieved from http://www.hongkiat.com/blog/desktop-customization-tools/

Foy, B. D. (2002). The Singleton Design Pattern, 19–36.

Gashti, S., Pujolle, G., & Rotrou, J. (2009). An UPnP-based context-aware framework for ubiquitous mesh home networks. *2009 IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*, 400–404. doi:10.1109/PIMRC.2009.5449966

Gui, F., Guillen, M., Rishe, N., Barreto, A., Andrian, J., & Adjouadi, M. (2009). A Client-Server Architecture for Context-Aware Search Application. *2009 International Conference on Network-Based Information Systems*, 539–546. doi:10.1109/NBiS.2009.75

Guiss, A. (2011). Como alterar a interface do Android. Retrieved from http://www.tecmundo.com.br/pdf/13210-como-alterar-a-interface-do-android-video-.pdf

Johnson, S. (2007). A framework for mobile context-aware applications. *BT Technology Journal*, *25*(2), 112–111. doi:10.1007/s10550-007-0033-5

Kobayashi, C. Y. (2004). A Tecnologia Bluetooth e aplicações. *Europe*, (2369461), 1–5.

Korhonen, K. (2011). Predicting mobile device battery life.

Kostakos, V., & O'Neill, E. (2007). NFC on Mobile Phones: Issues, Lessons and Future Research. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, (photo 5), 367–370. doi:10.1109/PERCOMW.2007.84

Kovács, L., Mátételki, P., & Pataki, B. (2009). Service-oriented Context-aware Framework. *Electronic Proceedings in Theoretical Computer Science*, *2*, 15–26. Retrieved from http://arxiv.org/abs/0906.3924

Lopes, J. L. B. (2006). Sensibilidade ao Contexto na computação Pervasiva : Avaliando o Uso de Ontologias.

Lopez, M. (2012). The Four Phases Of Enterprise Mobility. *Forbes*. Retrieved from http://www.forbes.com/sites/maribellopez/2012/09/27/the-four-phases-of-enterprise-mobility/

Norvig, P., & Stuart J, R. (1988). Inteligência Artificial Sumário. *Russell The Journal Of The Bertrand Russell Archives*, 498.

Offutt, J. (2010). The Java Beans Design Pattern, 1–13.

Padovitz, a., Loke, S. W., & Zaslavsky, a. (2008). Multiple-Agent Perspectives in Reasoning About Situations for Context-Aware Pervasive Computing Systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *38*(4), 729–742. doi:10.1109/TSMCA.2008.918589

Pplware. (2011). LauncherPro - O Android ainda mais elegante e moderno, 1–2. Retrieved from http://pplware.sapo.pt/smartphones/android/launcherpro-o-android-ainda-mais-elegante-e-morderno.pdf

Sang-Hak, L., & Tae-Choong, C. (2004). System Architecture for Context-Aware Home Application. *Software Technologies for Future Embedded and Ubiquitous Systems 2004 Proceedings Second IEEE Workshop on*, 149–153.

Sesterheim, G. C. (2011). Ambiente de configuração de alertas a usuários da plataforma Moodle para dispositivos móveis.

Silva, I. G. L., & Castro, J. F. B. (2005). Projeto e Implementação de Sistemas Multi-Agentes: O Caso Tropos (Presentation). *Universidade Federal de Pernambuco*.

Starck, D. (2010). Organize a sua Área de trabalho !, (1). Retrieved from http://www.tecmundo.com.br/pdf/3999-organize-a-sua-area-de-trabalho-.pdf

Theimer, M. M. (1994). Disseminating Active Map Information to Mobile Hosts. *Ieee Network*.

Vasile, C. (2011). LauncherPro 0.8.4 Available for Download , Adds Improvements for Gingerbread, (March), 2013. Retrieved from http://news.softpedia.com/newsPDF/LauncherPro-0-8-4-Available-for-Download-Adds-Improvements-for-Gingerbread-192034.pdf

Want, R., Schilit, B. N., Adams, N. I., Gold, R., Petersen, K., Goldberg, D., Ellis, J. R., et al. (1995). The ParcTab Ubiquitous Computing Experiment. (T. Imielinski & H. F. Korth, Eds.)*Mobile Computing*, *353*(CSL-95-1), 45–101. doi:10.1007/978-0-585-29603-6_2

Weerasinghe, T., & Warren, I. (2010). Odin: Context-Aware Middleware for Mobile Services. *2010 6th World Congress on Services*, 661–666. doi:10.1109/SERVICES.2010.69

Whitwam, R. (2011). LauncherPro vs ADWLauncher EX vs GO Launcher: What is the Best Home Replacement on Android? Retrieved from http://www.tested.com/tech/android/2621-launcherpro-vs-adwlauncher-ex-vs-go-launcher-what-is-the-best-home-replacement-on-android/

XCubeLabs. (2012). The Android History.

Xu, F. (2009). Observer Design Pattern, 1–8.

Yamin, A. C. (2004). Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas , Móveis e Conscientes do Contexto da Computação Pervasiva.

IEEE 802.11 website (2012). The Working Group Setting the Standards for Wireless LANs - http://www.ieee802.org/11/ - acessado em 21/12/2012.

Nakahati et. Al (2006). Padrões para redes sem fio.

Tecnologia – Android é utilizado por 43% dos americanos e é o OS líder - http://www.tecnologia.com.pt/2011/11/android-e-utilizado-por-43-dos-americanos-e-e-o-os-lider/

Google's Android Bluetooth API <http://developer.android.com/guide/topics/wireless/bluetooth.html>

Google's Android NFC API <http://developer.android.com/reference/android/nfc/package-summary.html>

Google's Android WiFi API <http://developer.android.com/reference/android/net/wifi/package-summary.html>

Google's Android Location API <http://developer.android.com/reference/android/location/package-summary.html>

The Official Bluetooth Site - http://www.bluetooth.com

SAP. *Standardized* **Technical Architecture Modeling: Conceptual and Design Level.** [S.l.]. 2007.