UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
COLEGIADO DE SISTEMAS DE INFORMAÇÃO

CAMOBILE: UM FRAMEWORK PARA SISTEMAS SENSÍVEIS AO CONTEXTO NA PLATAFORMA IOS

Matheus Matos de Farias

Salvador
2012

UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
COLEGIADO DE SISTEMAS DE INFORMAÇÃO

CAMOBILE: UM FRAMEWORK PARA SISTEMAS SENSÍVEIS AO CONTEXTO NA PLATAFORMA IOS

Matheus Matos de Farias

Monografia apresentada ao Curso de Graduação em Sistemas de Informação, Universidade do Estado da Bahia - UNEB, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação

Orientador:
Eduardo Manuel de Freitas Jorge

Salvador
2012

Folha de Aprovação

Monografia de Projeto Final de Graduação apresentada ao Curso de Graduação em Sistemas de Informação, Universidade do Estado da Bahia - UNEB com o título CA-Mobile: Um Framework para Sistemas Sensíveis ao Contexto na Plataforma iOS, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação, defendida por Matheus Matos de Farias e aprovada em 11 de Agosto de 2012 em Salvador, no estado da Bahia, pela comissão examinadora:

Prof. Eduardo Manuel de Freitas Jorge
Doutor em Difusão do Conhecimento
Universidade do Estado da Bahia
Orientador

Prof. Alexandre Rafael Lenz
Mestre em Informática
Universidade do Estado da Bahia

Prof. Uedson Santos Reis
Mestre em Modelagem Computacional
SENAI CIMATEC

Agradecimentos

Agradeço primeiramente aos meus pais pelas oportunidades que me foram dadas, pela educação diária e por todo o apoio que foi dado em todos os anos. Agradeço aos meus irmãos e a minha família por estarem ao meu lado e pelo carinho em todos os momentos de minha vida.

Agradeço a minha amiga Camila Rocha, por ter me ajudado nos momentos de dificuldade que encontrei no início da faculdade. Ao meu amigo Raul Lermen por ter me ensinado o que é gostar do que se aprende. A minha amiga Luciana Campos por ter me escutado em momentos que eu precisei. Também agradeço aos demais amigos que me ajudaram e me incentivaram.

Agradeço ao professor Eduardo Jorge por toda a orientação prestada durante o desenvolvimento deste trabalho. E por fim, agradeço a todos os professores que contribuíram na minha graduação.

Resumo

O objetivo deste trabalho é projetar uma solução computacional para apoiar o desenvolvedor na construção de aplicativos que utilizam o conceito de computação sensível ao contexto. Este conceito é uma das áreas de pesquisa da computação ubíqua, a qual surgiu da integração da computação móvel com a computação pervasiva. Através da computação sensível ao contexto é possível caracterizar as condições de um usuário no ambiente em que ele está através da coleta de informações por um dispositivo. Entretanto, o desenvolvimento de aplicativos sensíveis ao contexto não é algo trivial. Por isto, neste projeto foi construído um *framework* para a plataforma iOS, intitulado de CAMobile, o qual pode ser reutilizado na construção de novos aplicativos sensíveis ao contexto. Além disto, foram desenvolvidos dois aplicativos para validarem o *framework*.

Palavras chave: Computação Ubíqua, Computação Sensível ao Contexto, *Framework*, iOS, CAMobile.

Abstract

The objective of this work is to design a computacional solution to support the developer to build applications that use the concpet of context-sensitive computing. This concept is one of the research areas of ubiquitous computing, which emerged from the integration of mobile computing with persavise computing. Through context-sensitive computing is possible to characteriz the conditions of a user in the environment where he is by collecting information from a device. However, the development of context-aware applications is not trivial. Therefore, in this project was built a framework for the iOS platform, titled CAMobile, which can be reused in the construction of new context-sensitive applications. In addition, two applications were developed to validate the framework.

Keywords: Ubiquitous Computing, Context-Sensitive Computing, Framework, iOS, CAMobile.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O uso de dispositivos móveis tem crescido nos últimos anos, chegando, segundo a *Gartner* (GARTNER, 2011), a marca de 428,7 milhões de unidades no segundo trimestre de 2011. Apesar disto, ainda não é tão simples utilizar um dispositivo móvel devido as suas limitações, como tamanho da tela e de teclado. Por isto, a procura por aplicativos que facilitem o uso e automatizem tarefas também aumentou.

Para facilitar e automatizar tarefas nos dispositivos tem se aplicado o conceito de contexto de uso. Este conceito é uma importante área de pesquisa da computação ubíqua, que visa melhorar a experiência do usuário na utilização de dispositivos.

A ideia é colher informações, de forma imperceptível ao usuário, as quais caracterizariam o estado atual do mesmo no momento em que o dispositivo estiver sendo utilizado. Ao reconhecer o contexto, o dispositivo deve disponibilizar serviços que sejam de interesse naquele momento para assim melhorar a experiência de utilização.

Um exemplo desta aplicação é o aplicativo chamado "Lembretes", que vem incluso com a última versão do sistema operacional da *Apple*[1] para dispositivos móveis, o iOS 5. Este aplicativo permite organizar a vida de um usuário em uma lista de afazeres com datas e locais. Deste modo, o usuário pode ser lembrado de comprar leite quando for ao mercado (REMINDERS, 2012). Outro exemplo é o *Fast App Launching with Context* (FALCON), que é uma extensão do *Windows Phone OS*, o qual permite que aplicativos sejam pré-carregados em determinada localização, por exemplo, quando o usuário estiver em casa algum jogo é pré-carregado, ou quando o usuário estiver no escritório o aplicativo de *e-mail* é pré-carregado. Desta forma, os aplicativos podem ser rapidamente iniciados (FALCON, 2012).

Para colher informações é necessário que os aplicativos façam uso de sensores presentes

[1]Empresa multinacional norte-americana que tem o objetivo de projetar e comercializar produtos eletrônicos de consumo, *software* de computador e computadores pessoais.

nos dispositivos móveis, como *Global Positioning System* (GPS), acelerômetro[2], *bluetooth*. E este é o principal fator que dificulta o desenvolvimento de aplicações que utilizem contexto de uso. Isto porque o desenvolvedor, além de se preocupar com a lógica da sua aplicação, precisa se preocupar em como recuperar as informações e gerenciar os eventos de cada sensor, o que não é algo trivial.

Existem componentes que facilitam a obtenção de dados de contexto de uso, como localização geográfica, data e hora. Porém, estes dados ficam soltos para serem usados onde o desenvolvedor quiser e com isso, nem sempre é simples fazer destes dados um real contexto.

Também existem *frameworks* que auxiliam o desenvolvimento de sistemas sensíveis ao contexto, como o *Context Toolkit* (CONTEXTTOOLKIT, 2011) e o JCAF (*The Java Context-Awarenes Framework*) (JCAF, 2011). Entretanto, nenhum destes se preocupam inteiramente com as limitações dos dispositivos móveis.

Há um componente chamado *ContextMobile* (CONTEXTMOBILE, 2012) que tem o objetivo de simplificar a construção de sistemas sensíveis ao contexto na plataforma *Android*[3].

Na plataforma iOS há alguns *frameworks* disponibilizados pela própria *Apple* que auxiliam na obtenção de contextos de uso. Porém, para cada contexto é necessário utilizar um *framework* diferente, fazendo com que o desenvolvedor precise de uma grande curva de aprendizado para construir aplicativos sensíveis ao contexto.

No iOS existem poucos *frameworks* feitos por desenvolvedores comuns que venham ajudar a construção de aplicativos. A maioria dos desenvolvedores acabam utilizando os *frameworks* da própria *Apple*. No entanto, é interessante que mais *frameworks* sejam disponibilizados para este sistema, aumentando assim o desenvolvimento de aplicativos que necessitem de técnicas mais robustas.

Esta quantidade pequena de *frameworks* é consequência da comunidade de desenvolvedores desta plataforma ainda ser pequena, porém sua popularidade vem crescendo, como pode ser visto no site da *Tiobe* (TIOBE, 2012), onde é mostrado que a linguagem *Objective C* subiu três posições entre março de 2011 e março de 2012 em relação as outras linguagens de programação.

Como existe uma demanda por aplicativos sensíveis ao contexo, entretanto há uma

[2]Instrumento, incluso em alguns dispositivos móveis, capaz de medir a aceleração sobre objetos.
[3]Sistema operacional desenvolvido pela Google para dispositivos móveis.

certa dificuldade em desenvolvê-los, se torna interessante a construção de um framework que auxilie os desenvolvedores no desenvolvimento deste tipo de aplicativo. Por isto, o objetivo deste trabalho é projetar uma solução computacional composta pela especificação de um *framework* para aplicações sensíveis ao contexto na plataforma iOS. Dois exemplos de aplicativos serão construídos para validar a funcionalidade do *framework*. Estes aplicativos estarão manipulando contextos baseados em localização geográfica e contextos relacionados à velocidade.

A metodologia utilizada no desenvolvimento do projeto será a metodologia chamada de Projeto Dirigido por Exemplo. Onde primeiramente são analisados exemplos de aplicações de um domínio, extraindo semelhanças, para em seguida se construir uma hierarquia de classes genéricas. Por fim, são feitos testes desenvolvendo aplicações utilizando o *framework*, avaliando a sua aderência nas aplicações.

Este trabalho possui 9 capítulos. Sendo que o Capítulo 1 faz uma introdução do trabalho. No Capítulo 2 são apresentados conceitos da área de pesquisa que este trabalho está inserido. Já no Capítulo 3 é abordado fundamentos da Engenharia de *Software* que são aplicados no projeto. Enquanto, no Capítulo 4 é mostrado a metodologia utilizada com maiores detalhes, evidenciando os recursos, os experimentos e a validação do projeto. No Capítulo 5 é apresentado o CAMobile, mostrando alguns trabalhos relacionados, detalhando sua arquitetura, seu funcionamento, seu fluxo de execução e seu projeto de baixo nível. Já no Capítulo 6 são expostos os testes e resultados do CAMobile. No Capítulo **??** são mostrados os resultados obtidos com a monografia. Por fim, as considerações finais e sugestões de trabalhos futuros são apresentados no Capítulo 7.

2 VISÃO GERAL DA COMPUTAÇÃO UBÍQUA, COMPUTAÇÃO SENSÍVEL AO CONTEXTO E PLATAFORMA IOS

Neste Capítulo serão apresentados os principais conceitos da área de pesquisa que este trabalho está inserido. Inicialmente são explanados conceitos sobre a computação ubíqua, em seguida são abordados conceitos sobre computação sensível ao contexto, que é uma área de pesquisa da computação ubíqua. Por fim, é exposto um resumo da plataforma iOS, na qual o CAMobile está disponível.

2.1 Computação Ubíqua

A computação ubíqua é a integração entre a computação móvel e a computação pervasiva. A primeira surgiu como um paradigma no qual os usuários poderiam carregar seus computadores pessoais e manter certa conectividade com outras máquinas (COULOURIS; DOLLIMORE; KINDBERG, 2007). Já a segunda implica que o computador está embarcado no ambiente de forma invisível para o usuário. Desta forma, o computador tem a capacidade de obter e utilizar informações do ambiente para controlar, configurar e ajustar uma aplicação para melhor atender as necessidades do dispositivo ou do usuário.

Então, a partir da computação ubíqua, os dispositivos poderão construir dinamicamente modelos computacionais dos ambientes, nos quais um usuário interage, configurando seus serviços dependendo da necessidade.

A ideia básica da computação ubíqua é que a computação move-se para fora das estações de trabalho e computadores pessoais, tornando-se pervasiva em nossa vida co-

tidiana. Marc Weiser, considerado uma das maiores referências da computação ubíqua, vislumbrou há uma década atrás que, no futuro, computadores habitariam os mais triviais objetos: etiquetas de roupas, xícara de café, interruptores de luz, canetas, etc., de forma invisível para o usuário (ARAUJO, 2003).

Para acontecer o que Weiser vislumbrou é necessário que a informação possa ser acessada em qualquer lugar, através de vários dispositivos. Além disto, é necessário que estes dispositivos interajam entre si e com o ambiente, executando algumas tarefas de forma autônoma. Possibilitando, por exemplo, que um usuário ao fazer uma ligação via videoconferência em um celular, tenha o vídeo transferido para uma tela maior, como a de uma televisão, quando for percebido a presença desta tela no ambiente.

Existem pelo menos três princípios identificados na computação ubíqua que são: diversidade; descentralização e conectividade.

Os dispositivos ubíquos têm uma visão de funcionalidade como de propósito especifíco. Ou seja, vários dispositivos podem oferecer funcionalidades que se sobrepõem, porém um dispositivo que atende a uma necessidade específica pode ser mais apropriado para esta função do que o outro que atende a várias (ARAUJO, 2003). Por exemplo, os *tablets* possuem uma série de funcionalidades que auxiliam no dia a dia de um usuário, como checar e-mail, navegar na *web*, entre outros, porém não é o melhor dispositivo para fazer leitura de *ebooks*, como o *Kindle* (KINDLE, 2012), um leitor de livros digitais desenvolvido pela *Amazon* (AMAZON, 2012). Assim, através da computação ubíqua vão surgir diversos dispositivos e aplicações que atenderão às necessidades específicas dos usuários.

Na computação ubíqua não pode existir fronteiras na conectividade. Os dispositivos e suas aplicações se movem com o usuário de forma transparente, utilizando várias redes diferentes.

Por exemplo, suponha que todos os meios de exibição e escrita fixos de uma sala - quadros, livros, folhas de papel, notas adesivas, etc. - fossem substituídos por dezenas ou centenas de computadores individuais com telas eletrônicas (COULOURIS; DOLLIMORE; KINDBERG, 2007). Cada um destes componentes ajudariam um usuário em alguma tarefa ou função, mostrando assim a diversidade e descentralização da computação ubíqua, e cooperariam entre si para a construção de um ambiente inteligente, necessitando assim de conectividade.

Há vários desafios para serem superados na computação ubíqua nos níveis tecnológico, social e organizacional. No nível tecnológico há vários desafios ligados à evolução de

dispositivos, *softwares*, segurança, conectividade, recursos, adaptação e sincronismo.

No nível social há o problema da privacidade, onde os usuários podem desconfiar dos sistemas por causa de seus recursos de percepção. A presença de sensores nos espaços inteligentes significa que se torna possível rastrear os usuários eletronicamente, em uma escala potencialmente maciça e jamais vista (COULOURIS; DOLLIMORE; KINDBERG, 2007).

No nível organizacional é gerado novas preocupações, por exemplo, como o empregador supervisionará seus empregados, já que os computadores estarão por toda parte conectados uns aos outros por redes (ARAUJO, 2003).

Na computação ubíqua existem diversas áreas de pesquisa e uma das principais é a computação sensível ao contexto, a qual é um paradigma que visa desenvolver aplicações que obtenham proveito de informações de contexto, no momento em que um dispositivo esteja sendo utilizado.

2.2 Computação Sensível ao Contexto

A computação sensível ao contexto está fortemente ligada aos dispositivos móveis, devido à mobilidade dos usuários e objetos computacionais e pelo fato do usuário desejar informações em qualquer lugar, a qualquer momento, principalmente em movimento, fazendo com que o contexto do usuário esteja em constante modificação (LOUREIRO et al., 2009).

Na literatura são encontradas algumas definições para o termo computação sensível ao contexto, dentre elas:

- "O estudo de aplicações que se adaptam de acordo com a localização do usuário, grupo de pessoas, objetos próximos ao usuário e as mudanças ocorridas com esses objetos ao longo do tempo" citado por Schilit e Theimer (1994);
- "Aplicações que dinamicamente modificam ou adaptam seu comportamento baseado nas informações de contexto da aplicação ou do usuário" citado por Ryan, Pascoe e Morse (1997);
- "Sistema que utiliza informações relativas ao contexto para fornecer informações ou serviços relevantes ao usuário", citado por Dey e Abowd (2000).

O principal objetivo da computação sensível ao contexto é caracterizar as condições de um usuário no ambiente em que ele está, através da coleta de informações por um dispositivo móvel.

Estas informações seriam o contexto, o qual é qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade, a qual pode ser uma pessoa, um lugar ou um objeto, que seja considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação (DEY, 2001).

Então, um contexto compreende informações relevantes para um usuário em um dado momento, tais como: localização, ambiente que está inserido, energia, hora, dispositivos próximos, capacidade de processamento, entre outros.

Para um dispositivo reconhecer um contexto, ele precisa de sensores e controladores. Os sensores são dispositivos que medem parâmetros físicos e fornecem seus valores para o *software*. Inversamente, os controladores são dispositivos controlados pelo *software*, que afetam o mundo físico. Por exemplo, existem sensores que medem posição, orientação, carga e níveis de luz e som. Enquanto existem controladores para ar condicionado programável e para motores (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Um exemplo de dispositivo que tem a capacidade de reconhecer contextos de uso é o *iPhone*, que é um dispositivo que executa sobre a plataforma iOS e possui alguns sensores, como o de localização, através do GPS e de movimento, através do acelerômetro.

2.3 Plataforma iOS

O iOS é o sistema operacional da *Apple*, baseado em Unix[1], que roda em dispositivos como *iPhone*, *iPod Touch* e *iPad*. Este sistema controla o *hardware* dos dispositivos e disponibiliza as tecnologias necessárias para implementar aplicações nativas (APPLE, 2011).

A arquitetura do iOS é similar à arquitetura básica do *Mac OS X*[2]. De modo que, o sistema atua como um intermediário entre o *hardware* e as aplicações. A arquitetura é dividida em camadas, onde no nível mais baixo se encontram os serviços e tecnologias fundamentais de todas as aplicações, enquanto no nível mais alto se encontram serviços e

[1]Sistema operacional portátil, multitarefa e multiusuário de propriedade do The Open Group, um consórcio formado por empresas de informática.

[2]Sistema operacional proprietário baseado no kernel Unix, desenvolvido, fabricado e vendido pela *Apple*.

tecnologias mais sofisticados. A Figura 2.1 mostra as camadas do IOS.

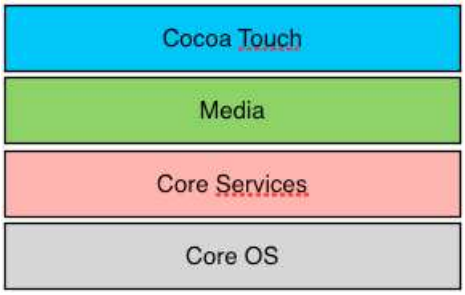

Figura 2.1: Camadas da plataforma iOS adaptada de APPLE (2011).

Estado	**Descrição**
Não Executando	O aplicativo ainda não foi iniciado ou foi executado, mas foi terminado pelo sistema.
Inativo	O aplicativo está executando, mas não está respondendo a eventos. Um aplicativo usualmente fica neste estado por um curto período quando ele está transitando para um estado diferente.
Ativo	O aplicativo está executando e respondendo a eventos.
Background	O aplicativo está em *background* e executando algum código. Alguns aplicativo entram neste estado rapidamente antes de serem suspensos. Contudo, um aplicativo que requisita um tempo extra de execução pode ficar neste estado por mais tempo.
Suspenso	O aplicativo está em *background* mas não está executando código. O sistema move aplicativos para este estado automaticamente e não o notifica antes de fazer isto. Enquanto está suspenso, um aplicativo continua em memória mas não executa nenhum código. Quando uma condição de pouca memória ocorre, o sistema pode finalizar aplicativos suspensos para ganhar mais memória para aplicativos que estejam sendo executadas.

Tabela 2.1: Estados dos aplicativos no iOS.

A camada *Cocoa Touch* cuida da parte visível do usuário, ou seja a interface que vai aparecer na tela. Também controla a interação do usuário com a tela através de toque, do acelerômetro ou do uso das câmeras.

Esta camada é a mais usada pelo desenvolvedor, pois nela se encontram os principais *frameworks* que são utilizados na construção de aplicativos, dando suporte para tecnologias como multi-tarefa, *touch-based input*, *push notification* entre outros serviços do sistema de alto nível.

A camada *Media* é responsável pelos recursos de multimídia dos dispositivos, tais como os sons, as imagens e vídeos que são reproduzidos.

A camada *Core Services* gerencia os serviços do dispositivo como as ligações e os

envios de mensagem de texto, assim como serviços de protocolos de comunicação de rede e do banco de dados.

A camada *Core OS* é o nível mais baixo do sistema e por isto é a mais básica. Ela controla o funcionamento do sistema operacional, gerenciando a memória, a bateria, a luminosidade da tela, a segurança, entre outras funções.

A *Apple* trouxe o recurso de multitarefa para o iOS em sua quarta versão. Permitindo assim que aplicativos executassem tarefas específicas em segundo plano enquanto o usuário usa outros aplicativos. Entretanto o funcionamento da multitarefa neste sistema é um pouco diferente.

Em qualquer sistema multitarefa, a *Central Processing Unit* (CPU) chaveia de programa para programa, executando cada um deles por dezenas ou centenas de milissegundos. Estritamente falando, enquanto a cada instante a CPU executa somente um programa, no decorrer de um segundo ela pode trabalhar sobre vários programas, dando aos usuários a ilusão de paralelismo. Algumas vezes, nesse contexto, fala-se de pseudoparalelismo, para contrastar com o verdadeiro paralelismo de *hardware* dos sistemas multiprocessadores (TANENBAUM, 2010).

Ou seja, o sistema operacional executa diversas tarefas em um determinado tempo, alternando cada processo a cada instante.

Para entender melhor a diferença do funcionamento da multitarefa no iOS são mostrados na Tabela 2.1 os estados dos aplicativos no iOS.

Quando um aplicativo no estado 'Ativo' é interrompido devido a uma ligação ou porque o usuário apertou a tecla *'Home'* do dispositivo, o aplicativo passa para o estado *'Background'* e tem apenas cinco segundos para salvar dados e informações da interface para quando o aplicativo voltar ao estado 'Ativo' estar tudo como antes. Feito isto, o estado passa para 'Suspenso' (APPLE, 2011).

Entretanto, é possível que um aplicativo peça ao iOS por mais tempo para executar algumas tarefas. Mas, esta execução poderá levar no máximo dez minutos, passado este tempo o sistema operacional muda o estado para 'Suspenso'.

Além disto, há cinco tipos de aplicativos que podem ficar executando em *background* por um longo período de tempo sem que o sistema operacional o suspenda. A Tabela 2.2 possui a descrição da *Apple* para estes tipos.

O desenvolvedor precisa informar ao iOS que o aplicativo é de um destes tipos. Isto

é feito através da declaração da variável *'UIBackgroundModes'* no arquivo '*Info.plist*'.

Então, a multitarefa no iOS é limitada. Os aplicativos que não são dos tipos citados na Tabela 2.2 possuem pouco período de tempo para executarem quando estão no estado *'Background'*. E desta forma, as tarefas executadas, em grande parte, são de aplicativos que estão no estado 'Ativo'.

Tipo	**Descrição**
Áudio	Aplicativos que tocam conteúdo em formato de áudio para o usuário enquanto está em *background*.
Localização	Aplicativos que mantém os usuários informados de sua localização.
Voip	Aplicativos que provêem o serviço de fazer ligações usando a conexão da internet.
Conteúdo Externo	Aplicativos que fazem downloads e processam conteúdo de revistas e jornais em *background*.
Acessório Externo	Aplicativos que trabalham com algum acessório externo que precisam se comunicar em um certo período.

Tabela 2.2: Tipos de aplicativos que tem permissão de executar por um longo período em *background*.

3 ENGENHARIA DE SOFTWARE

Neste Capítulo são apresentados os fundamentos da engenharia de *software* que são aplicados no projeto do *framework* desenvolvido neste trabalho. Inicialmente é mostrado um resumo do objetivo da engenharia de *software*. Em seguida são abordados alguns conceitos de reúso de *software*. Por fim, são apresentados o *framework* e seu desenvolvimento.

3.1 Visão Geral

A Engenharia de *Software* atua na aplicação de abordagens sistemáticas ao desenvolvimento e manutenção de *software*. Os objetivos principais da Engenharia de *Software* são a melhora da qualidade do *software* e o aumento da produtividade da atividade de desenvolvimento de *software* (FAIRLEY, 1985). A reutilização de *software*, em contraposição ao desenvolvimento de todas as partes de um sistema, é um fator que pode levar ao aumento da qualidade e da produtividade da atividade de desenvolvimento de *software*.

3.2 Reúso de *Software*

A engenharia de *software* baseada em reúso é uma estratégia da engenharia em que o processo de desenvolvimento é orientado para o reúso de *softwares* existentes. A mudança para o desenvolvimento baseado em reúso foi uma resposta às exigências de menores custos de produção e manutenção de *software*, entregas mais rápidas de sistemas e *softwares* de maior qualidade. De forma que se aumente o retorno sobre os investimentos em *software* (SOMMERVILLE, 2011).

O reúso de artefatos de *softwares* já desenvolvidos e depurados, reduz o tempo de desenvolvimento de *software*, de testes e as possibilidades de introdução de erros na produção

de novos artefatos (SILVA, 2000). E com isso, há um aumento na qualidade e na produtividade no desenvolvimento de *software*.

Sommerville (2011) lista uma série de benefícios e problemas do reúso de *software*, os quais são mostrados nas Tabelas 3.1 e 3.2, respectivamente.

Benefício	**Explicação**
Redução dos custos	Menos componentes precisam ser especificados, concebidos, implementados e validados.
Confiança aumentada	Os *softwares* reusados, experimentados e testados em sistemas em funcionamento provavelmente são mais confiáveis do que um novo *software*.
Risco de processo reduzido	O custo do *software* existente já é conhecido, considerando que os custos de desenvolvimento são sempre uma questão que envolve julgamento. Esse é um importante fator para o gerenciamento de projeto, pois reduz a margem de erro de estimativa de custos de projeto, o que é particularmente verdadeiro quando componentes de *software* relativamente grandes, como subsistemas, são reusados.
Uso eficaz de especialistas	Em vez de repetir o mesmo trabalho, especialistas em aplicações podem desenvolver *softwares* reusáveis que encapsulem seu conhecimento.
Conformidade com padrões	Alguns padrões, como os de interface de usuário, podem ser implementados como um conjunto de componentes reusáveis. O uso de interfaces de usuário-padrão melhora a confiança, pois os usuários cometem menos erros quando são apresentados a interface familiares.
Desenvolvimento acelerado	Muitas vezes, os custos gerais de desenvolvimento não são tão importante quanto entregar um sistema ao mercado o mais rápido possível. O reúso de um *software* pode acelerar a produção do sistema, pois pode reduzir o tempo de desenvolvimento e validação.

Tabela 3.1: Benefícios do reúso de *software*.

Para se ter uma melhor eficiência com o reúso de *software*, visto seus benefícios e problemas, é necessário que os processos de desenvolvimento de *software*, desde a fase de requisitos e planejamento sejam adaptados.

Existem várias formas de reutilização de artefatos de *software*, que vão desde o nível de código, onde trechos de código já desenvolvidos são reutilizados, até os níveis de análise e projeto.

A orientação a objetos fornece funcionalidades para que classes possam ser reutilizadas, bem como métodos por meio de seus mecanismos de herança e polimorfismo. Componentes de *software* definem unidades reutilizáveis que oferecem serviços através de interfaces bem definidas. Padrões de projeto é outra abordagem mais abstrata que

objetiva a reutilização de projetos de soluções para problemas recorrentes. Além destas formas de reutilização, a tecnologia de *frameworks* possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura os conceitos mais gerais da família de aplicações (BARRETO, 2006).

Tipo	**Descrição**
Maiores custos de manutenção	Se o código-fonte de um sistema ou um componente de *software* reusáveis não estiverem disponíveis, os custos de manutenção podem ser maiores, pois os elementos reusados do sistema podem tornar-se cada vez mais incompatíveis com as alterações do sistema.
Falta de ferramentas de suporte	Algumas ferramentas de *software* não suportam o desenvolvimento com reúso. Pode ser difícil ou impossível integrar essas ferramentas com um sistema de biblioteca de componentes. O processo de *software* assumido por essas ferramentas pode não considerar o reúso.
Síndrome de 'não-inventado-aqui'	Alguns engenheiros de *software* preferem reescrever componentes, pois acreditam poder melhorá-los. Isso tem a ver, parcialmente com aumentar a confiança e, parcialmente com o fato de que escrever *softwares* originais é considerado mias desafiador do que reusar *softwares* de outras pessoas.
Criação, manutenção e uso de uma biblioteca de componentes	Preencher uma biblioteca de componentes reusáveis e garantir que desenvolvedores de *software* possam utilizar essa biblioteca podem ser ações caras. Processos de desenvolvimento precisam ser adaptados para garantir que a biblioteca seja usada.
Encontrar, compreender e adaptar os componentes reusáveis	Componentes de *software* precisam ser descobertos em uma biblioteca, compreendidos e, às vezes, adaptados para trabalhar em um novo ambiente. Os engenheiro precisam estar confiantes que encontrarão, na biblioteca, um componente, antes de incluírem a pesquisa de componente como parte de seu processo normal de desenvolvimento.

Tabela 3.2: Problemas com reúso.

3.3 *Framework*

Várias definições de *frameworks* são encontradas tanto na literatura como em artigos e teses acadêmicas.Algumas são descritas a seguir:

- Larman (2005) apresenta um *framework* como um conjunto de interfaces e classes que colaboram para fornecer serviços para a parte básica e constante de um sistema lógico.
- Sommerville (2011) diz que um *framework* é uma estrutura genérica estendida para se criar uma aplicação ou subsistema mais específico.

- Silva (2000) define *framework* como uma estrutura de classes interrelacionadas, que correspondem a uma implementação incompleta para um conjunto de aplicações de um domínio. Sendo que esta estrutura de classes deve ser adaptada para a geração de aplicações específicas.

Uma das definições descritas por Barreto (2006) diz que um *framework* é definido como um *software* parcialmente completo projetado para ser instanciado. O *framework* define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também são explicitados os lugares ou pontos de extensão (*hot-spots*[1]) nos quais devem ser feitas adaptações do código para um funcionamento específico de certos módulos.

Existem muitas outras definições diferentes de *frameworks*, porém elas não são contraditórias a essas que foram apresentadas.

Os *frameworks* fornecem suporte para recursos genéricos, suscetíveis de serem usados em todas as aplicações de tipos semelhantes. Por exemplo, um *framework* de interface de usuário fornecerá suporte para tratamento de eventos de interface e incluirá um conjunto de recursos que possam ser usados para construir *displays*. Então, o desenvolvedor fica responsável por especializá-los, adicionando funcionalidade específica para uma aplicação específica. Por exemplo, em um *framework* de interface de usuário, o desenvolvedor define *layouts* de *display* apropriados para a aplicação que está sendo implementada (SOMMERVILLE, 2011).

Portanto, um *framework* tem o objetivo de gerar diferentes aplicações dentro de um domínio. Sendo que ele possui classes abstratas que servem para representar partes de um sistema conceitualmente, podendo ser adaptadas às necessidades de uma aplicação específica.

Um *framework* é caracterizado por três aspectos. O primeiro é que os *frameworks* fornecem infraestrutura e projeto, desta forma é reduzida a quantidade de código a ser desenvolvido, testado e depurado. O desenvolvedor não possui a responsabilidade de estabelecer a arquitetura da aplicação, ficando responsável apenas por estender ou particularizar o comportamento do *framework*, de forma a moldá-lo a uma necessidade específica (SILVA, 2000).

[1]Hotspots são partes nos quais os programadores que usam o framework adicionam o seu código para especificar uma funcionalidade de sua aplicação.

O segundo é seu grau de reutilização, que é superior ao grau de reúso promovido tanto pelo reúso de rotinas, que aparece em bibliotecas de funções, quanto pelo reúso de classes, que possibilitam reutilização de atributos, além da reutilização de rotinas. Isto, porque o *framework* reusa um conjunto de classes interligadas ao invés de isoladas. Além disto, o *framework* possui uma característica chamada de "inversão de controle", onde os métodos especializados pelo desenvolvedor são chamados dentro do *framework* ao invés de serem chamado no código da aplicação.

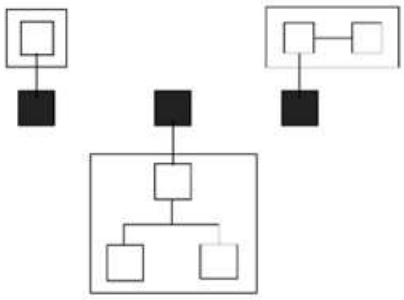

Figura 3.1: Aplicação desenvolvida reutilizando classes e rotinas.

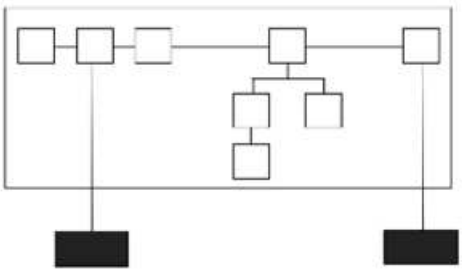

Figura 3.2: Aplicação desenvolvida utilizando um *framework*.

As figuras 3.1 e 3.2 mostram a diferença em relação ao controle. Os objetos em preto representam os componentes implementados pelo desenvolvedor, enquanto os objetos em branco representam objetos que são reutilizados.

Na Figura 3.1 o componente reutilizado complementa o componente que é implementado. Já na Figura 3.2 o componente implementado completa os componentes que são reutilizados.

Pela Figura 3.2 também é possível observar que o *framework* impõe uma arquitetura, onde os objetos se relacionam e onde novas implementações devem se adaptar. Desta forma, o *framework* promove reúso de código, análise e design, sendo que o reúso de análise e projeto vem do fato de que, um *framework* representa um projeto de um sistema abstrato, analisado anteriormente e projetado para capturar todas as características

comuns de um domínio de problema.

Os *frameworks* podem ser classificados quanto ao seu escopo em *frameworks* de infraestrutura de sistemas, *frameworks* de integração de *middleware*[2] e *frameworks* de aplicações corporativas.

A Tabela 3.3 mostra as definições destas classificações por Sommerville (2011).

Classicação	**Descrição**
Frameworks de infraestrutura de sistemas	Esses *frameworks* apoiam o desenvolvimento de infraestruturas de sistema, como comunicações, interfaces de usuário e compiladores.
Frameworks de integração de *middleware*	Trata-se de um conjunto de normas e classes de objetos associados que oferecem suporte a componentes de comunicação e troca de informações. Exemplos desse tipo de *frameworks* incluem o .NET da *Microsoft* e o *Enterprise Java Beans* (EJB).
Frameworks de aplicações corporativas	Esses estão relacionados com domínios de aplicação específicos, como telecomunicações ou sistemas financeiros. Eles incorporam conhecimentos sobre domínios de aplicações e apoiam o desenvolvimento de aplicações de usuário final.

Tabela 3.3: Classificações de *frameworks* quanto ao escopo.

Além de serem classificados quanto ao escopo, *frameworks* também possuem classificação quanto à forma usada para estendê-los. A Tabela 3.4 mostra as definições destas classificações por Barreto (2006).

Frameworks caixa branca exigem que o desenvolvedor possua um bom conhecimento sobre sua estrutura interna e por isso, há uma maior dificuldade em seu uso. Enquanto,

[2]Programa que faz a mediação entre um software e demais aplicações.

Classicação	**Descrição**
Framework caixa branca (*white box*)	São instanciados usando herança, através da implementação do Padrão de projeto *Template Method*, métodos abstratos que são implementados nas subclasses, ou da definição de métodos ganchos (*hook methods*), métodos com uma implementação padrão que pode ser redefinida na subclasse.
Framework caixa preta (*black box*)	São instanciados através de configurações e composições, através da definição de classes que implementam uma determinada interface ou contrato. Os ponto de extensão dos *frameworks* caixa preta frequentemente são definidos seguindo o padrão de projeto *Strategy*.
Framework caixa cinza (*gray box*)	Possuem as características conjuntas dos *frameworks* caixa branca e preta, de forma a tentar evitar as desvantagens dos dois.

Tabela 3.4: Classificações de *frameworks* quanto à forma de estendê-lo.

os *frameworks* caixa preta não exigem tanto conhecimento da estrutura interna do *framework*, no entanto tem uma flexibilidade menor.

Portanto, *frameworks* são estruturas de classes interrelacionadas, que permitem não apenas reutilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações, por terem a arquitetura do sistema definido, liberando o desenvolvedor de *software* desta preocupação (SILVA, 2000).

Os *frameworks* são uma abordagem eficaz de reúso, mas são caros para serem introduzidos em processos de desenvolvimento de *software*. Eles são inerentemente complexos e pode demorar meses para alguém aprender a usá-los. Pode ser difícil e caro avaliar *frameworks* disponíveis para a escolha do *framework* mais adequado. A depuração de aplicações baseadas em *framework* é difícil, pois o desenvolvedor pode não compreender como os métodos de um *framework* interagem (SOMMERVILLE, 2011).

3.4 Desenvolvimento de *Frameworks*

O desenvolvimento de um *framework* corresponde a uma evolução iterativa de sua estrutura de classes. Este refinamento cíclico também é motivado pela busca de adaptação de estrutura de classes, aos aspectos de generalidade e flexibilidade, e implica no manuseio de uma grande quantidade de informações. Assim, a adoção de mecanismos de descrição que apóiem o processo de desenvolvimento pode diminuir a dificuldade de desenvolver *frameworks*, bem como produzir um registro do projeto útil pra a utilização e a manutenção de *frameworks* (SILVA, 2000).

A característica principal que se procura no desenvolvimento de um *framework* é a generalidade em relação a conceitos e funcionalidades do domínio tratado. Também é importante que a estrutura produzida seja flexível, apresentando característica de alterabilidade e extensibilidade.

Alterabilidade reflete a capacidade do *framework* alterar suas funcionalidades, em função da necessidade de uma aplicação específica, o que é operacionalizado através de uma identificação adequada das partes da estrutura que diferem em aplicações distintas do mesmo domínio. Já a extensibilidade se refere à manutenbilidade do *framework*. Um *framework* possui uma estrutura de classes mais complexa que a estrutura de uma aplicação do seu domínio. Esta estrutura é construída em ciclos iterativos. A evolução da estrutura do *framework* se estende por toda a sua vida útil, pois à medida em que é utilizado, novos recursos podem ser agregados. Assim, na definição de abstrações, deve-se ter

a preocupação em prever futuras utilizações para o *framework*, inclusive a possibilidade de estender os limites do domínio tratado (SILVA, 2000).

Em termos práticos, dotar um *framework* de generalidade, alterabilidade e extensibilidade requer uma cuidadosa identificação das partes que devem ser mantidas flexíveis e a seleção de soluções de projeto de modo a produzir uma arquitetura bem estruturada. Isto passa pela observação de princípios de projeto orientado a objetos, como o uso de herança para reutilização de interfaces (ao invés do uso de herança para reutilização de código); reutilização de código através de composição de objetos; preocupação em promover polimorfismo, na definição de classes e métodos, de modo a possibilitar acoplamento dinâmico etc. (JOHNSON; FOOTE, 1988).

O ciclo de vida de um *framework* difere do ciclo de vida de aplicações convencionais, isto porque o *framework* depende das aplicações que são geradas por ele. A Figura 3.3 ilustra o ciclo de vida de um *framework*.

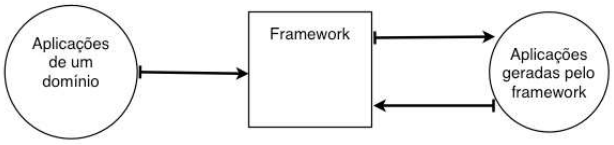

Figura 3.3: Ciclo de Vida de um *Framework*.

O ciclo de vida se inicia através de uma análise de domínio em cima de aplicações deste domínio tratado. O desenvolvedor analisa as aplicações e desenvolve o *framework* generalizando as diferentes estruturas das aplicações. Depois que o *framework* é criado, são desenvolvidas novas aplicações utilizando o novo *framework* e então é analisado o que está faltando e o que pode ser alterado, gerando em seguida novas versões do *framework* com as alterações.

Existem algumas metodologias de desenvolvimento de *software*, entre elas o Projeto Dirigido por Exemplo e o Projeto Dirigido por Hot Spot. Estas metodologias se caracterizam por estabelecer o processo de desenvolvimento de *frameworks* em linhas gerais, sem se ater à definição de técnicas de modelagem ou detalhar o processo (SILVA, 2000). Ou seja, elas definem uma série de procedimentos que incluem a captura de informações de um domínio, a construção e o teste da estrutura de *frameworks*.

4 METODOLOGIA

Neste Capítulo é apresentada a metodologia aplicada na construção do projeto. Detalhando quais os recursos são utilizados, quais os experimentos foram feitos e quais os critérios de validação do projeto.

O desenvolvimento de um *framework* para o domínio de aplicação é decorrente de um processo de aprendizado a respeito deste domínio, que se processa concretamente a partir do desenvolvimento de aplicações ou do estudo de aplicações desenvolvidas. Porque as pessoas pensam de forma concreta, e não de forma abstrata, a abstração do domínio, que é o próprio *framework*, é obtida a partir da generalização de casos concretos - as aplicações. Abstrações são obtidas de uma forma *bottom-up*, a partir do exame de exemplos concretos: aspectos semelhantes de diferentes aplicações podem dar origem a classes abstratas que agrupam as semelhanças, cabendo às classes concretas do nível hierárquico inferior a especialização para satisfazer cada caso (JOHNSON, 1993).

Uma metodologia que segue este conceito é a chamada por Projeto Dirigido por Exemplo, por isto e também por ser uma metodologia simples, ela foi a escolhida para o desenvolvimento do projeto.

O processo de desenvolvimento de um *framework* segundo o Projeto Dirigido por Exemplo possui três etapas: análise, projeto e teste.

A etapa de análise compreende a análise do domínio, onde são assimiladas as abstrações já conhecidas, é coletado exemplos de programas que poderiam ser desenvolvidos a partir do *framework* e é avaliado a adequação de cada exemplo.

A etapa de projeto compreende a projeção de uma hierarquia de classes que possa ser especializada para abranger os exemplos (SILVA, 2000).

E a etapa de teste compreende ao teste do *framework*, desenvolvendo exemplos utilizando-o e avaliando o *framework* desenvolvido.

4.1 Recursos

Na realização deste projeto foi utilizado um *MacBook Pro* 13" com processador *Intel Core i5 Dual Core* 2,4 GHZ e memória de 4 GB para o desenvolvimento do serviço/*framework* e de pequenos aplicativos que validaram a solução. Também foi utilizado um *Ipod Touch* de 64 GB e um *iPhone* para realização de demonstrações e testes.

O sistema operacional utilizado pelo MacBook é o OS X 10.7 Lion, enquanto o sistema operacional do Ipod Touch e Iphone é o iOS 5.0.1. A IDE de desenvolvimento foi o Xcode 4.2. Também foi utilizado o iOS Simulator para fazer simulações no próprio MacBook.

4.2 Experimentos

Inicialmente foram feitos alguns exemplos de aplicações que possuem o domínio de contexto de uso para auxiliar no desenvolvimento do CAMobile.

Primeiramente foi construído um aplicativo que usa o contexto de uso de data e hora. O aplicativo é da categoria *TODO List*, também conhecida como Lista de Tarefas. Um usuário pode cadastrar tarefas que serão lembradas, através de uma notificação em determinado horário e data.

Em seguida foi construído um aplicativo que usa o contexto de localização geográfica. Onde o usuário é notificado caso esteja em uma determinada localização.

Depois disto, foi feita uma evolução do segundo aplicativo, dando duas novas opções para o usuário ser notificado. Primeiro quando estiver, em determinada data ou hora. E segundo quando estiver em determinada localização e em determinada data ou hora.

Através do desenvolvimento destes aplicativos, foi feito uma análise do projeto deles, procurando quais são os pontos semelhantes e o que poderia ser abstraído na construção do *framework*.

Após a construção do *framework*, foram desenvolvidos dois aplicativos para dispositivos que possuem o sistema iOS com o intuito de testar o *framework* e avaliar o seu uso. Os aplicativos atenderam aos cenários de uso descritos nas seções 4.2.1 e 4.2.2.

4.2.1 Promoções em um Shopping

Produtos em oferta fazem as vendas de uma loja crescer, porque as pessoas tendem a aproveitar as promoções para economizarem.

Quando uma pessoa vai em um shopping, muitas vezes não tem conhecimento de todas as promoções que existem no dia. Isto acontece, pois um shopping possui muitas lojas e as pessoas nem sempre tem tempo suficiente de visitar todas elas.

Desta forma, foi feito um aplicativo que tem como objetivo mostrar para os usuários as promoções dentro de um shopping. Quando o usuário entra no shopping o aplicativo verifica em um servidor se alguma loja possui uma promoção disponível, caso exista, uma lista de promoções é mostrada para o usuário. Além disto, quando é um dia de liquidação no shopping, uma notificação é lançada para o usuário, informando sobre a liquidação.

Através deste aplicativo um shopping pode negociar com os proprietários das lojas o anúncio das promoções e também espaços de publicidade.

4.2.2 Velocidade do carro

Em muitos computadores de bordo presentes em carros existe uma funcionalidade, que alerta o motorista, quando o mesmo ultrapassa certa velocidade dirigindo o veículo.

Ao se adicionar este computador no carro, o seu preço se eleva muito, além de que alguns carros não possuem nem a opção de adicionar o computador. Seria interessante então, uma forma mais simples de sinalizar o motorista quando o mesmo ultrapassar certa velocidade.

Desta forma, será feito um aplicativo que tem a finalidade de apresentar uma notificação ao usuário, chamando a atenção do motorista, informando para ter cuidado com a velocidade, além de executar um som de alerta, quando uma velocidade pré-determinada for ultrapassada.

4.3 Critérios de Validação

Para este trabalho estar completo e validado é necessário que os dois aplicativos experimentais que utilizarão o *framework* desenvolvido estejam funcionando corretamente.

5 CAMOBILE

Neste capítulo é apresentado o CAMobile, evidenciando alguns trabalhos relacionados, descrevendo os objetivos do *framework*, sua arquitetura em alguns níveis, seu funcionamento e sua adaptação na plataforma iOS. Também é mostrado seu fluxo de execução e seu projeto de baixo nível, contendo um diagrama de classes e alguns diagramas de sequência relacionados ao *framework*.

5.1 Trabalhos Relacionados

Nesta Seção serão mostrados alguns trabalhos já realizados com o objetivo de diminuir a complexidade no desenvolvimento de sistemas sensíveis ao contexto, além de um comparativo entre estes trabalhos e o CAMobile.

5.1.1 *Context Toolkit*

Context Toolkit é um conjunto de ferramentas desenvolvido pelo *Georgia Institute of Technology* no ano 2000. Ele foi bastante utilizado para a construção de protótipos de aplicações sensíveis ao contexto.

O *Context Toolkit* utiliza o conceito de *widgets* para representar contextos. Um *widget* é um componente de interface gráfica do usuário (GUI). Para se construir estes *widgets* é utilizada a linguagem de marcação *Extensible Markup Language* (XML) (CONTEXT-TOOLKIT, 2011).

O principal desafio a ser superado pelo *Context Toolkit* é o tratamento do contexto. Não há um serviço central que trate os contextos, e por isto é necessário criar um serviço para cada *widget* que represente um contexto.

5.1.2 JCAF

JCAF é um *framework* criado para dar suporte a Computação Sensível ao Contexto e tem o objetivo de projetar aplicações sensíveis ao contexto.

O JCAF utiliza a linguagem *Java* e possui um modelo de programação genérico, extensível e expressivo para desenvolver sistemas sensíveis ao contexto e modelos de contexto (JCAF, 2011).

Um diferencial do JCAF é a sua infraestrutura orientada a serviço, onde existem vários serviços que se comunicam entre si e dividem a responsabilidade de adquirir, gerenciar e distribuir os contextos de uso (JCAF, 2011).

Para monitorar um contexto é preciso modelar primeiro uma tupla contendo informações de uma entidade, um relacionamento e um item. E depois submeter a um dos serviços passando esta tupla e um evento.

5.1.3 *ContextMobile*

O *ContextMobile* foi desenvolvido por Diogo Monte e Gabriel Brawne em 2011. Ele se propõe em trazer simplicidade no desenvolvimento de aplicativos para plataforma Android que utilizem contexto de uso.

O *ContextMobile* disponibiliza serviços de contexto de uso para serem reutilizados por outras aplicações. Um destes serviços consiste em obter a localização do usuário através do GPS presente em um dispositivo móvel. Este componente também propõe um serviço único que monitore a existência e obtenção de contextos.

Para monitorar um contexto, é preciso solicitar a monitoração deste contexto ao *ContextMobile*, que informará ao aplicativo quando o contexto for identificado.

5.1.4 Comparativo

Na Tabela 5.1 é mostrado um comparativo entre o CAMobile e os trabalhos relacionados.

O CAMobile se aproxima mais do *ContextMobile*, pois os dois se tratam de frameworks disponíveis para plataformas móveis e também porque a interface que gerencia os controladores do CAMobile é parecido com o serviço centralizado do *ContextMobile*. O CAMobile também é semelhante ao JCAF, pois apresenta um conjunto de controladores

que controlam os sensores, semelhante ao conjunto de serviços do JCAF.

5.2 Objetivos do CAMobile

O CAMobile visa reduzir a complexidade do desenvolvimento de aplicações sensíveis ao contexto na plataforma iOS. De modo que, um desenvolvedor possa reutilizar componentes não precisando conhecer aspectos técnicos de forma completa relacionados ao gerenciamento de sensores como GPS e Acelerômetro, que são utilizados para a obtenção e manipulação de um contexto de uso.

A partir do CAMobile, um desenvolvedor poderá programar a ocorrência de eventos após a verificação da existência de um contexto. Como por exemplo, uma notificação ser lançada para o usuário após ser identificado que o mesmo se encontra em uma localização específica.

Este *framework* possibilita a conjugação de contextos de uso, possibilitando, por exemplo, que um evento ocorra caso seja identificado mais de um contexto.

5.3 Arquitetura

A Figura 5.1 mostra a arquitetura em seu nível mais alto.

Nesta arquitetura é mostrado o CAMobile contendo um componente chamado Contexto e algumas aplicações que se comunicam com o *framework*.

Framework **Característica**	**Context Toolkit**	**JCAF**	**Context Mobile**	**CAMobile**
Ano	2000	2009	2011	2012
Linguagem	XML	*Java*	*Java*	*Objective-C*
Plataforma	*Desktop*	Multiplataforma	*Android*	iOS
Arquitetura	*Widgets*	Conjunto de serviços	Serviço centralizado	Diversos controladores gerenciados por uma interface
Funcionamento	Uso de *widgets*	Passagem de tuplas e eventos para os serviços	Solicitação de monitoração de contexto ao *ContextMobile*	Solicitação a uma interface para executar um evento após a identificação de um contexto

Tabela 5.1: Comparativo entre o CAMobile e seus trabalhos relacionados.

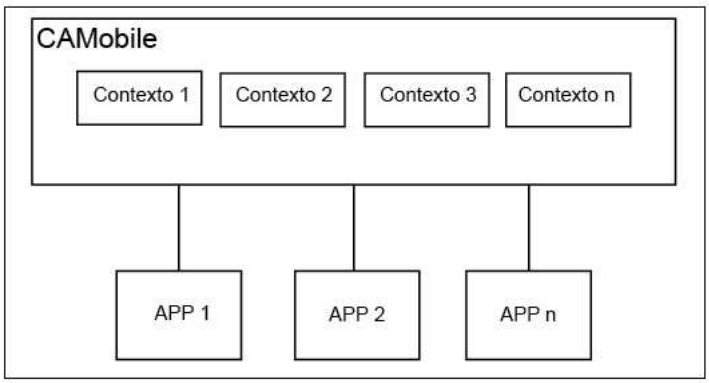

Figura 5.1: Arquitetura de alto nível.

O CAMobile engloba o gerenciamento de contextos, como por exemplo, localização, data/hora, acelerômetro, bluetooth, entre outros. Este gerenciamento é feito pelo componente Contexto, que são pequenos blocos de códigos responsáveis por "observar" os sensores do dispositivo (câmera, relógio, acelerômetro, giroscópio, etc.) e notificar ao *framework* quando as condições dos sensores atenderem sua solicitação. Cada contexto é implementado para observar um único sensor e seu principal objetivo é isolar os demais componentes do acesso ao sensor, assumindo o papel de interface entre eles. Diversos contextos podem ser acoplados ao *framework*, cada um responsável por monitorar um sensor do dispositivo, sendo limitado apenas pelos recursos de *hardware* disponíveis no dispositivo.

Desta forma, uma aplicação que necessita da informação da data para lançar uma notificação para o usuário se comunicaria com o *framework* solicitando ser notificada quando uma determinada data for identificada. O *framework* por sua vez, solicitaria ao componente Contexto responsável por monitorar data observar quando a data passada pela aplicação fosse identificada.

A Figura 5.2 mostra a arquitetura do *framework* com um pouco mais de detalhes, apresentando a mesma com um novo componente denominado Interface.

A interface tem como principal função, fornecer um padrão de comunicação entre as aplicações e o CAMobile. Desta forma, para uma aplicação solicitar a observação de um determinado contexto, se faz necessário o uso da interface. Da mesma forma, o *framework* utiliza a Interface para notificar a aplicação que um determinado contexto foi identificado.

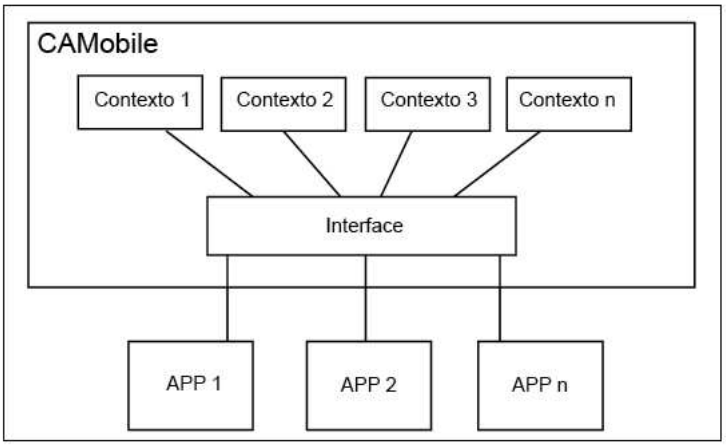

Figura 5.2: Arquitetura com interface.

5.4 Funcionamento do *Framework* CAMobile

Nesta Seção é explicado o funcionamento do CAMobile , trazendo a forma que uma aplicação interage com o *framework*.

Uma aplicação, ao se conectar com o *framework*, e solicitar a observação de um evento, precisa seguir um padrão de comunicação definido pelo componente Interface. Esse padrão define que as seguintes informações devem ser passadas pelas aplicações:

- Evento: Uma ação que será executada, após a identificação de um contexto. Por exemplo, uma aplicação, que deseja alterar uma *label* da tela, após identificar que está em determinada localização, precisa informar ao *framework* a ação de alterar a *label*.

  Este Evento segue o Padrão de Projeto *Command*, o qual tem a intenção de encapsular uma solicitação como um objeto, e com isso permitindo parametrizar clientes com diferentes solicitações (GAMMA et al., 2000).

- Condições do Evento: As condições do evento são os contextos que devem existir para que o evento ocorra, no exemplo descrito acima, seria passado a localização.

Após identificar as condições dos eventos solicitados, o *framework* iniciará o gerenciamento dos contextos que ainda não estiverem em execução, sendo responsável por finalizá-los quando não houver mais aplicações interessadas neles. Dessa forma, não haverá consumo de energia desnecessário.

Para gerenciar e manipular os contextos são utilizados controladores específicos para cada contexto. Estes controladores utilizam *frameworks* disponibilizados pela *Apple* ou

seguem o Padrão de Projeto *Observer* que, segundo Gamma et al. (2000), define uma dependência um-para-muitos entre objetos, de forma que quando um objeto mudar de estado, todos os seus dependentes sejam notificados e/ou atualizados automaticamente.

Uma aplicação pode estar interessada em mais de um contexto. Neste caso em especial, a partir da agregação de dois ou mais contextos simples, surgirá um contexto composto. Essa agregação nada mais é que uma combinação lógica entre as condições do evento utilizando as cláusulas *"and"* ou *"or"*. Por exemplo, uma aplicação solicita os contextos de data/hora e de GPS, e deseja ser notificada todos os dias, às 08h00min, se o aparelho estiver em determinado local.

Ao receber esta solicitação, o *framework* irá instanciar os controladores dos contextos de data/hora e de GPS, caso ainda não estejam instanciados, e apenas notificará a aplicação, quando as duas condições (quando for 08h00min e estiver no local especificado) forem satisfeitas, caso a combinação lógica utilizada seja *"and"*. Ou notificará a aplicação assim que uma das condições (ou quando for 08h00min ou quando estiver no local especificado) for satisfeita, caso a combinação lógica seja *"or"*. A aplicação deverá informar ao *framework* qual a condição lógica deverá ser utilizada quando mais de um evento for observado.

5.5 *Framework* no iOS

Inicialmente foi cogitada a idéia de desenvolver um serviço que fosse acoplado a arquitetura do *framework*. Sendo que este serviço executaria o tempo todo em *background* como uma única instância, monitorando a existência de contextos de uso, caso uma ou mais aplicações solicitassem.

Para o desenvolvimento da aplicação só seria necessário a adição de uma interface de comunicação com este serviço, enquanto este serviço ficaria responsável por se comunicar com os sensores do dispositivo.

Este serviço traria economia de recursos e bateria porque apenas ele se comunicaria com os sensores e não vários aplicativos ao mesmo tempo.

No entanto como foi visto na Seção 2.3, o conceito de multitarefa é um pouco diferente no iOS. Um aplicativo pode ficar no máximo 10 minutos executando em *background*, a não ser que faça parte de um dos tipos mostrados na Tabela 2.2.

Como um serviço que fique monitorando a existência de contextos de uso necessita de

mais de dez minutos executando em *background* e também não se enquadrando em nenhum dos tipos permitidos para executar em um longo período de tempo, fica inapropriada a criação de um serviço deste tipo para esta plataforma.

A falta do serviço não será um problema em relação ao consumo de recursos e bateria. Isto porque o iOS já gerencia alguns de seus sensores, como o sensor de GPS e o sensor de data e hora.

Como não há um serviço, então o *framework* e a comunicação com os sensores precisam ficar acoplados no aplicativo. A Figura 5.3 mostra a arquitetura do *framework* no iOS dentro de um aplicativo.

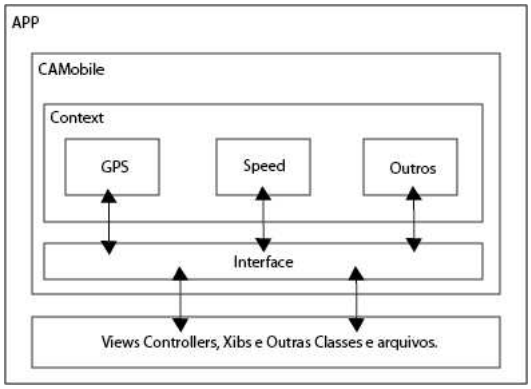

Figura 5.3: Arquitetura do *Framework* no iOS

Na Figura 5.3 é mostrado que um aplicativo é composto por classes, *controllers*, *xibs* (além de outros arquivos que o compõem) e pelo *framework*, o qual possuem caixas que são os contextos de uso.

O *framework* funcionará como uma interface entre as classes do aplicativo e sensores do dispositivos que tragam informações de contexto como o GPS, relógio (data/hora), entre outros.

Cada caixa que representa um contexto de uso utilizará dos *frameworks* do iOS para obter informações do contexto de uso. Por exemplo, a caixa de GPS vai utilizar o *framework CoreLocation* do iOS para obter informações de localização como latitude e longitude.

A utilização dos sensores que trazem informações de contexto de uso será simplificada para um desenvolvedor de sistemas sensíveis ao contexto. Isto porque o *framework* terá métodos que facilitaram a verificação de determinados contextos de uso. Podendo inclusive, deixar o aplicativo programado para um evento ser executado quando um ou mais contextos de uso (em conjunto) sejam encontrados.

Como foi visto na Tabela 2.2, aplicativos que mantém os usuários informados de sua localização possuem permissão de executarem por um longo período em *background*. Como localização é um dos contextos que foi implementado neste trabalho, então foi aproveitado seu fluxo de execução para acionar a monitoração dos outros contextos que foram implementados. Simplificando assim os testes dos outros contextos, porém trazendo a limitação de o controlador de localização sempre ser iniciado.

5.6 Fluxo de Execução do *Framework*

A Figura 5.4 mostra o fluxo de execução do CAMobile junto a uma aplicação em seu nível mais alto.

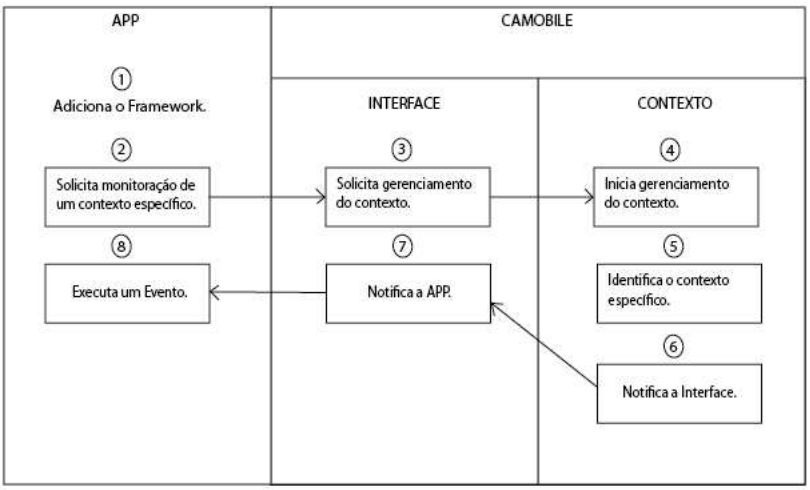

Figura 5.4: Fluxo de Execução em alto nível.

A Figura 5.4 é apresentada dividida entre um APP e o CAMobile, o qual está divido em dois componentes, a Interface e o componente Contexto, que chamaremos aqui de Controlador de Contexto. O fluxo mostra que são necessários pelo menos 8 passos para realizar a função principal do *framework*, que é executar uma ação após a identificação de um determinado contexto. Sendo que 3 passos são feitos na aplicação, enquanto 5 são feitos pelo *framework*.

Inicialmente o desenvolvedor precisa adicionar o *framework* em sua aplicação e em seguida solicitar ao CAMobile, através da Interface, a monitoração de um determinado contexto (ex.: localização). Feito isto, a Interface terá o papel de solicitar que o contexto determinado seja gerenciado. Após a solicitação, o componente Controlador de Contexto iniciará o gerenciamento, observando os sensores do dispositivo.

Quando o contexto determinado for identificado o componente Controlador de Con-

texto notificará a Interface que, por usa vez, notificará a aplicação que, enfim executará um Evento, que possui uma ação.

5.7 Projeto de Baixo Nível do *Framework*

A Figura 5.5 apresenta o diagrama de classes do CAMobile.

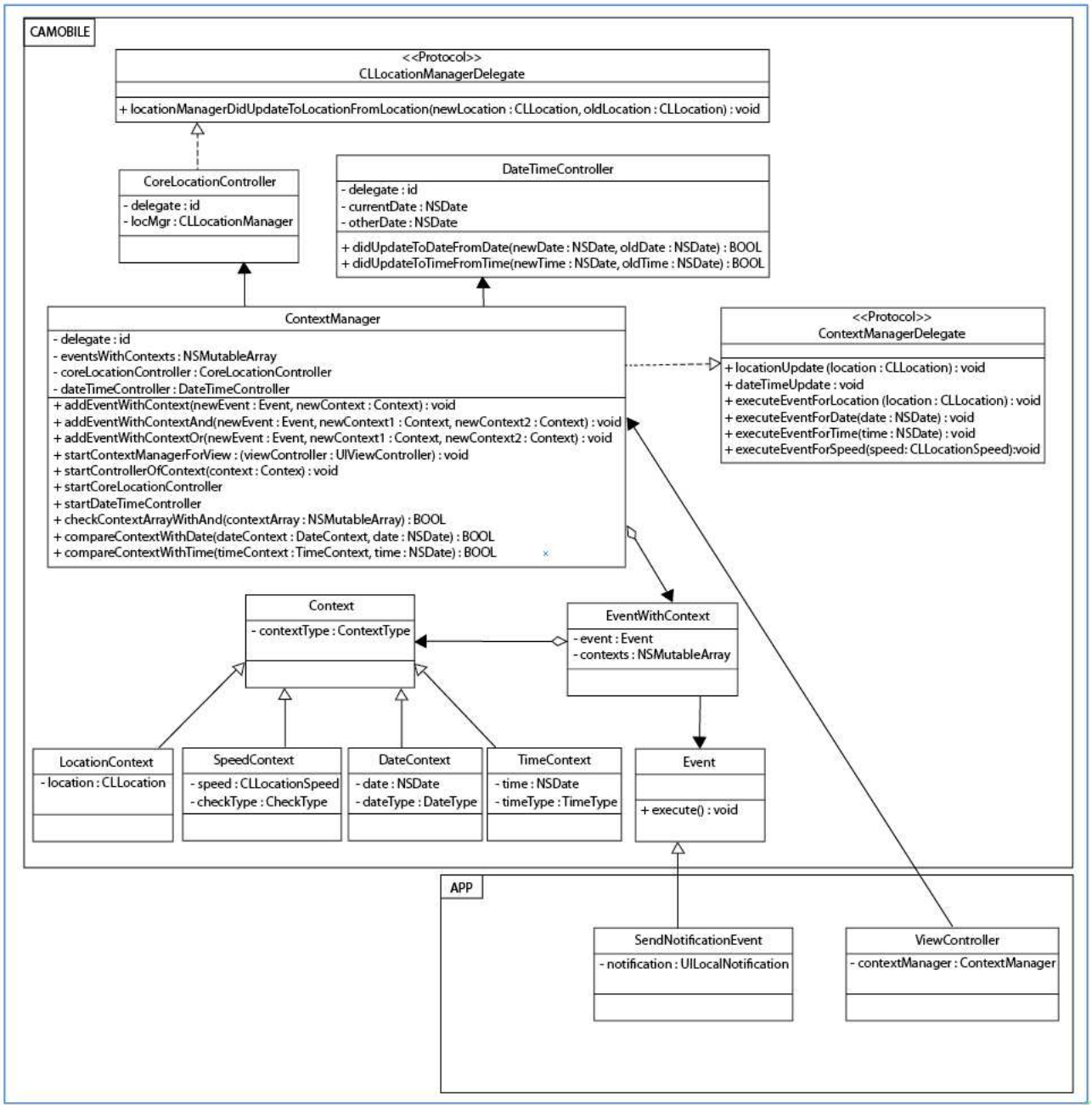

Figura 5.5: Diagrama de Classes

No diagrama de classes da Figura 5.5 é mostrado a classe *Context* e suas extensões: *LocationContext* (Localização), *SpeedContext* (Velocidade), *DateContext* (Data) e *Time-*

Context (Hora). Essas extensões representam os contextos que o *framework* consegue gerenciar em seu estado atual.

No diagrama também é mostrada a classe *Event*, que utiliza o padrão de projeto *Command*, o qual encapsula uma requisição através de um objeto. Os desenvolvedores terão que criar extensões de *Event*, para encapsular uma ação que será chamada quando for identificado um determinado contexto. Um exemplo desta extensão é a classe *SendNotificationEvent*, que encapsula o lançamento de uma notificação para o usuário.

A classe *ContextManager* representa o componente Interface da arquitetura do CAMobile é através dela que a aplicação se comunica com o *framework*, solicitando que um evento ocorra após a detecção de um contexto. Isto pode ser feito através do métodos *"addEventWithContext"*, *"addEventWithContextAnd"* e *"addEventWithContextOr"*.

Então, o *ContextManager* é uma interface de nível mais alto que torna o *framework* mais fácil de ser usado. E isto, segundo Gamma et al. (2000) é exatamente o que o Padrão de Projeto Façade se propõe.

As classes *CoreLocationController* e *DateTimeController* representam o componente Contexto da arquitetura do CAMobile. Elas gerenciam um contexto específico, *CoreLocationController* gerencia os contextos de Localização e Velocidade, enquanto *DateTimeController* controla o contexto de Data/Hora.

Já na Figura 5.6 apresenta-se o diagrama de sequência da execução de uma aplicação que estende o CAMobile. O diagrama se inicia com o desenvolvedor adicionando o *framework* em sua aplicação e termina com o CAMobile informando a aplicação da identificação de um contexto.

Para se utilizar o *framework* é necessário algumas configurações que são mostradas no tutorial de configuração do CAMobile presente no Anexo A.

O fluxo se inicia com o desenvolvedor adicionando o CAMobile, o que faz com que o *framework* seja empacotado na aplicação. Em seguida, é iniciada a implementação da aplicação, com a adição de um objeto da classe *ContextManager* e o desenvolvimento das ações da aplicação com o CAMobile, que seria a chamada dos métodos que iniciam o *ContextManager* e os *Controllers*. Também é implementada a ação de adicionar um Evento com Contexto, que pode ser chamado em algum método do ciclo de vida do *framework* do iOS ou em algum método que realize uma interação com o usuário. Quando o Contexto é identificado o *framework* notifica a aplicação.

Após a finalização da implementação um usuário pode utilizar a aplicação, e o fluxo

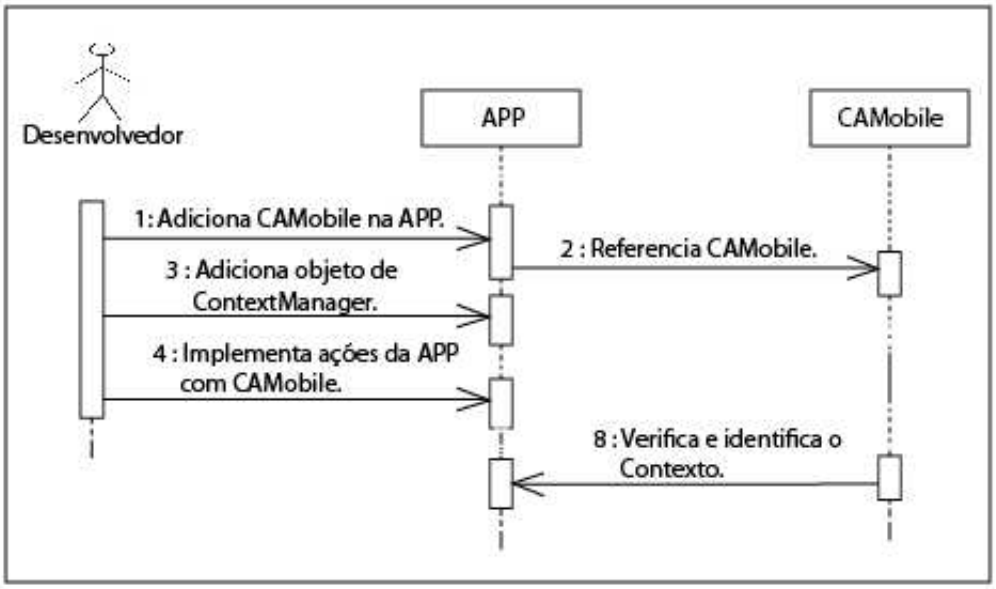

Figura 5.6: Diagrama de sequência do desenvolvedor.

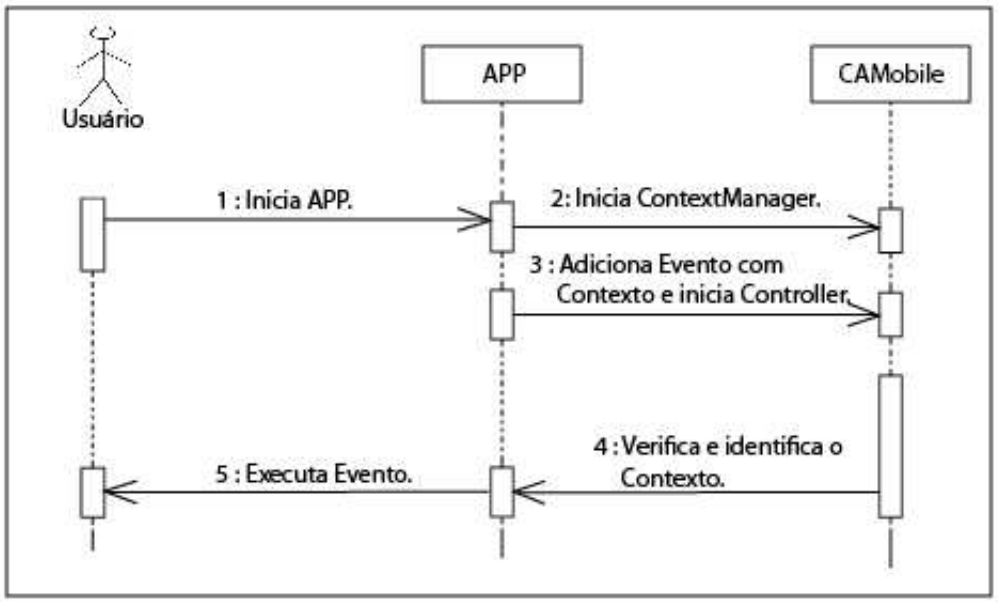

Figura 5.7: Diagrama de sequência do usuário.

deste uso está ilustrado na Figura 5.7.

O fluxo se inicia com o usuário iniciando a aplicação que chama um método que inicia o *ContextManager*. Feito isto, o usuário em algum momento fará alguma ação na aplicação que chamará um dos métodos que adicionam um Evento com um ou mais Contextos. Quando realiza esta operação o CAMobile verifica se o *Controller* do Contexto que vai ser adicionado já esta iniciado, caso ainda não esteja, o inicia.

Depois de adicionado um Evento com Contexto, o CAMobile passa a verificar cons-

tantemente a existência do contexto determinado. E quando este contexto é identificado a aplicação executa o Evento e mostra a ação para o usuário.

Na Figura 5.8 é mostrado o diagrama de sequência dos métodos que a aplicação tira proveito do CAMobile, utilizando como exemplo de *Controller* o *CoreLocationController*.

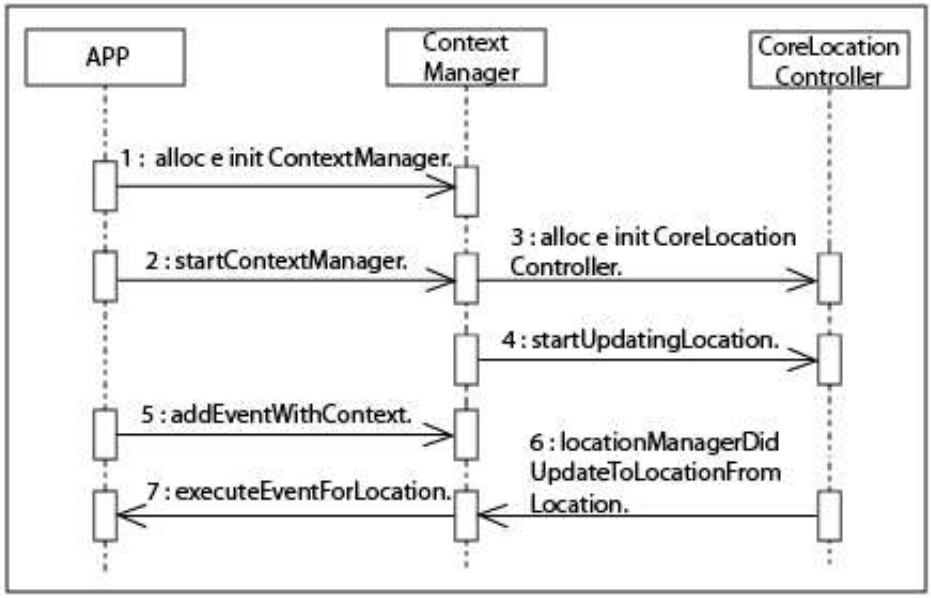

Figura 5.8: Diagrama de sequência dos métodos do CAMobile.

O fluxo se inicia quando a aplicação aloca e inicia o objeto do *ContextManager* e em seguida chama o método *"startContextManager"*, que por sua vez aloca e inicia um objeto do CoreLocationController e chama o método *"startUpdatingLocation"*, o qual faz com que o Controller observe mudanças de localização.

Em um outro momento a aplicação chama o método *"addEventWithContext"*, que adicionará eventos com contextos que serão verificados. Para verificar a existência deste contexto, o *CoreLocationController* executa o método *"locationManagerDidUpdateToLocationFromLocation"*, o qual chama o método *"executeEventForLocation"* do *ContextManager*. Neste momento o *ContextManager* verifica se o contexto é um dos que foram adicionados, caso seja o Evento associado é executado.

6 TESTES E RESULTADOS

Neste Capítulo será exposto como foram desenvolvidos os experimentos deste projeto. Inicialmente serão mostrados os primeiros aplicativos que ajudaram na construção do *framework*. E em seguida os dois aplicativos que validaram o projeto.

6.1 Primeiros Aplicativos

O primeiro aplicativo construído foi o que possui a categoria *TODO List*. Onde o usuário cadastra uma lista de tarefas para serem executadas em uma determinada data/hora e é notificado quando for esta data/hora.

O desenvolvimento foi simples, pois o iOS traz uma certa facilidade na criação de notificações. Sendo que o próprio sistema operacional gerencia o momento que a notificação será lançada. No Algoritmo 6.1 é mostrado a criação da notificação.

```
1  - (void)addNotification {
2      // Cria a notificacao .
3      UILocalNotification *localNotification = [[UILocalNotification alloc] init];
4      // Define quando a notificacao sera lancada .
5      localNotification.fireDate = self.datePicker.date;
6      // Define qual a mensagem apresentada na notificacao .
7      localNotification.alertBody = self.messageField.text;
8      localNotification.soundName = UILocalNotificationDefaultSoundName;
9      localNotification.applicationIconBadgeNumber = 1;
10     // Programa que a notificacao seja lancada pelo iOS .
11     [[UIApplication sharedApplication] scheduleLocalNotification: localNotification
12     [localNotification release];
13     [self.delegate dismissSetNotificationViewController];
14 }
```

Algoritmo 6.1: Código Fonte Criando Notificação

Na linha 3 do código fonte da criação da notificação é mostrado como a notificação é criada. Na linha 5 e 7 é mostrado como é definido quando a notificação será lançada e a mensagem que será exibida respectivamente. Já na linha 12 é solicitado ao iOS gerenciar esta notificação.

O segundo aplicativo construído teve o objetivo de notificar o usuário caso ele estivesse em determinada localização. O desenvolvimento deste aplicativo não foi tão simples, pois foi necessário conhecer o *Core Location Framework*, que é um *framework* disponibilizado pela *Apple* para gerenciar o GPS.

Foi percebido que o uso do *Core Location Framework* realmente não é algo trivial, pois é preciso conhecer tópicos avançados do iOS, como *Protocols* e *Delegates*, algo que normalmente é apenas apresentado nos livros de introdução ao desenvolvimento no iOS.

Em seguida, foi evoluído o segundo aplicativo, possibilitando ao usuário duas novas opções para ser notificado. Primeiro quando estiver, em determinada data ou hora. E segundo quando estiver em determinada localização e em determinada data ou hora.

A primeira nova opção adicionada no segundo aplicativo poderia ser feita, utilizando o método addNotifcation mostrado no Algoritmo 6.1. Entretanto, a segunda nova opção adicionada não seria possível fazer deste modo, pois o iOS gerencia apenas a data e hora, e não a localização. Tendo esta dificuldade, seria necessário algo que unisse a verificação de data/hora e da localização. A solução provida foi a criação de uma interface que gerenciasse um controlador para cada contexto. E com isso, já dando início ao desenvolvimento do CAMobile.

Foi visto que os aplicativos tinham em comum o gerenciamento dos contextos, então isto foi levado para o *framework* como controladores de contexto. Além disto, foi criada a interface que poderia gerenciar os controladores. Nesta interface também foram feitos métodos que facilitassem ao desenvolvedor solicitar o gerenciamento de um ou mais contextos.

6.2 Aplicativos de Validação

O primeiro aplicativo desenvolvido para validar o CAMobile foi o *iSaleMall*, um aplicativo que tem objetivo de mostrar as promoções de um determinado Shopping, quando o usuário chegar nele.

Desta forma é necessário que o aplicativo faça a ação de mostrar as promoções quando

for identificado que o usuário está no shopping. Isto foi feito utilizando o método "*addEvent: (Event *) newEvent withContext: (Context *) newContext;*" que é provido pela classe *ContextManager* do CAMobile. Na linha 8 do Algoritmo 6.2 é mostrada a chamada deste método no *iSaleMall*, onde é passado um objeto da classe *Event*, o qual possui encapsulado a ação de mostrar as promoções, e um objeto de *LocationContext*, que possui a informação da localização do shopping.

```
1  - (IBAction)changeSalvadorSwitch {
2      if (salvadorSwitch.on) {
3          LocationContext *locationContext = [[LocationContext alloc] init];
4          locationContext.location = [[CLLocation alloc] initWithLatitude:+55.75578600 longitude
                   :+37.61763300];
5          ShoppingEvent *shoppingEvent = [ShoppingEvent alloc];
6          shoppingEvent.view = self;
7          shoppingEvent.novoTexto = @"Salvador Shopping";
8          [contextManager addEvent:shoppingEvent withContext:locationContext];
9      }else{
10             // remove  o  evento .
11     }
12 }
```

Algoritmo 6.2: Código Fonte da chamada do método que adiciona um evento com contexto no iSaleMall.

Na Figura 6.1 é mostrada a tela inicial do aplicativo, onde um usuário pode ativar a exibição de promoções de um determinado shopping. Enquanto, na Figura 6.2 é mostrado o aplicativo exibindo as promoções.

O segundo aplicativo desenvolvido para validar o CAMobile foi o *BipSpeed*, um aplicativo que tem objetivo de notificar um motorista através de uma notificação e um som de alerta, quando for ultrapassada certa velocidade pré-determinada.

Desta forma é necessário que o aplicativo faça a ação de mostrar a notificação e execute o som de alerta quando for identificado que o usuário está numa velocidade superior com a que foi informada. Isto também foi feito utilizando o método "*addEvent: (Event *) newEvent withContext: (Context *) newContext;*". Na linha 11 do Algoritmo 6.3 é mostrada a chamada do método no *BipSpeed*, onde é passado um objeto da classe *Event*, o qual possui encapsulado a ação de mostrar a notificação e executar o som, e um objeto de *SpeedContext*, que possui a informação da velocidade.

Figura 6.1: Tela inicial do iSaleMall.

Figura 6.2: iSaleMall mostrando promoções.

```
1  - (IBAction)setarVelocidade {
2      NSString *velocidade = [[NSString alloc] initWithString:[velocidadeTextField text ]];
3      double vel = [velocidade doubleValue];
4      speed = vel / 3.6;  //    km / h  para  m / s
5      SpeedContext *speedContext = [[SpeedContext alloc] init];
6      speedContext.checkType = CheckType_Maior;
7      speedContext.speed = speed;
8      SendNotificationEvent *sendNotification = [SendNotificationEvent alloc];
9      sendNotification. notification  = [[NSClassFromString(@"UILocalNotification") alloc] init];
10     sendNotification.reminder = [[NSString alloc] initWithString:@"Cuidado com a velocidade!"];
11     [contextManager addEvent:sendNotification withContext:speedContext];
12 }
```

Algoritmo 6.3: Código Fonte da chamada do método que adiciona um evento com contexto no *BipSpeed*.

Na Figura 6.3 é mostrada a tela inicial do aplicativo, onde um usuário pode determinar uma velocidade. Enquanto, na Figura 6.4 é mostrado o alerta que é exibido após a velocidade determinada ser ultrapassada.

Figura 6.3: Tela inicial do *BipSpeed*.

Figura 6.4: BipSpeed mostrando alerta.

6.3 Resultados

A partir da construção dos primeiros aplicativos foi constatado que gerenciar e manipular os sensores dos dispositivos realmente não é algo trivial no iOS. O desenvolvedor

precisa conhecer aspectos mais avançados como *Protocols* e *Delegates*, além de conhecer os *frameworks* que manipulam os sensores, como o *Core Location Framework*, que manipula o GPS.

Já na construção dos aplicativos de validação foi notada uma certa simplicidade em manipular estes sensores, pois o desenvolver só fica responsável por fazer a solicitação do gerenciamento do contexto através de um método. Enquanto, o CAMobile cuida do gerenciamento dos sensores.

Então, o CAMobile trouxe simplicidade no desenvolvimento de aplicativos que utilizam contexto de uso, pois o desenvolvedor não precisa se preocupar com a manipulação dos sensores do dispositivo, podendo focar na lógica da aplicação em si.

Além da simplicidade no desenvolvimento de aplicativos sensíveis ao contexto, o CAMobile também possibilita reúso de *software*, porque rotinas que verificam e identificam contextos de uso são encapsuladas no *framework*. Toda vez que é necessário verificar a existência de um determinado contexto de uso é utilizada da mesma rotina para isto.

7 CONSIDERAÇÕES FINAIS

Muitos aplicativos que utilizam Contexto de uso têm surgido. Este aplicativos tem o intuito de colher informações do ambiente em que o usuário está inserido, de forma imperceptível, através dos dispositivos e com isso automatizar algumas tarefas. Esta automatização é demandada devido a limitações, como tamanho da tela e teclado, presentes nos dispositivos móveis, que dificultam o seu uso.

Entretanto, o desenvolvimento destes aplicativos ainda não é algo trivial, porque os desenvolvedores precisam manipular e gerenciar sensores do dispositivo, como GPS e acelerômetro, algo que possui certa complexidade.

Com o objetivo de diminuir a dificuldade no desenvolvimento destes aplicativos foi construído o CAMobile, um *framework* para sistemas sensíveis ao contexto para a plataforma iOS. Através deste *framework*, os desenvolvedores podem solicitar que um evento ou ação ocorra, caso algumas condições, que são os contextos de uso, sejam atendidas.

Primeiramente, o *framework* foi especificado com a construção de sua arquitetura, do diagrama de classes e de alguns diagramas de sequência. Também foi descrito o funcionamento e o fluxo de execução. Um artigo, que mostrou a arquitetura do CAMobile em alto nível foi publicado no *The 12th International Conference on Computational Science and Its Applications* (ICCSA 2012) (JORGE et al., 2012).

Em seguida, o *framework* foi implementado e depois foram desenvolvidos dois aplicativos que reutilizaram classes do CAMobile, com o propósito de validar o *framework*. Um se chama *iSaleMall* e utiliza o contexto de localização, enquanto o outro se chama *BipSpeed* e utiliza o contexto de velocidade. Estes aplicativos podem ser evoluídos e distribuídos para a comunidade.

Com o desenvolvimento destes aplicativos ficou constatado que o CAMobile atendeu ao seu objetivo proposto. Porque o *framework* possibilita reúso de software e traz simplicidade ao desenvolvimento de aplicativos que utilizam contexto de uso.

No entanto, no decorrer da construção do *framework* foi percebido que a plataforma iOS possui certas limitações ligadas ao que pode ficar executando em *background*, que dificultam uma melhor utilização do CAMobile. Isto pode ser melhorado em trabalhos futuros, caso a plataforma possibilite que mais aplicativos possam executar em *background*.

Além desta sugestão de trabalho futuro, também pode ser trazido para o *framework* novas formas de conjugar contextos, como por exemplo a cláusula "entre", onde um evento pode ser acionado caso o usuário esteja entre uma data inicial e uma data final por exemplo.

Outro trabalho futuro é a criação de novos controladores de contexto, trazendo assim mais poder ao CAMobile, possibilitando que mais contextos possam ser verificados. Por fim, o CAMobile pode ser portado para outras plataformas, como o *Windows Phone OS*[1] da *Microsoft* e o *Android* do *Google*.

[1]Sistema operacional para dispositivos móveis desenvolvido pela Microsoft.

REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON. Amazon.com. 2012. Acesso em: 28 jun. 2012. Disponível em: <http://www.amazon.com/>.

APPLE. Ios developer library: ios technology overview. 2011. Acesso em: 22 nov. 2011. Disponível em: <http://developer.apple.com/>.

ARAUJO, R. B. Computação ubíqua: Princípios, tecnologia e desafios. *23º Simpósio Brasilerio de Redes de Computadores*, 2003.

BARRETO, C. G. J. *Agregando Frameworks de Infra-Estrutura em uma Arquitetura Baseada em Componentes*. Dissertação (Mestrado) — PUC-Rio, Rio de Janeiro, 2006.

CONTEXTMOBILE. contextmobile. 2012. Acesso em: 05 mar. 2012. Disponível em: <https://github.com/contextmobile/Context-Mobile>.

CONTEXTTOOLKIT. Context toolkit. 2011. Acesso em: 11 nov. 2011. Disponível em: <http://www.contexttoolkit.org/>.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos: Conceitos e Projeto*. [S.l.]: Bookman, 2007.

DEY, A.; ABOWD, G. Towards a better understanding of context and context awareness. *Workshop on The What, Who, Where, When, and How of Context-Awareness*, 2000.

DEY, A. K. Understanding and using context. *Personal Ubiquitous Computing - Springer-Verlag*, v. 5, p. 4–7, 2001.

FAIRLEY, R. *Software Engineering concepts*. [S.l.]: McGraw-Hill,, 1985.

FALCON. Fast app launching with context. 2012. Acesso em: 25 mai. 2012. Disponível em: <http://people.cs.umass.edu/ dganesan/papers/MobiSys12-Falcon.pdf>.

GAMMA, E. et al. *Padrões de Projeto: Soluções de software orientado a objetos*. São Paulo: Bookman, 2000.

GARTNER. Technology research. 2011. Acesso em: 12 dez. 2011. Disponível em: <http://www.gartner.com/>.

JCAF. Jcaf - the java context-awareness framework. 2011. Acesso em: 11 dez. 2011. Disponível em: <http://www.daimi.au.dk/ bardram/jcaf/>.

JOHNSON, R. E. How to design frameworks. 1993. Acesso em: 03 mar. 2012. Disponível em: <st-www.cs.illinois.edu/users/johnson/cs497/notes98/day18.ps>.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Jounal of Object-Oriented Programming*, p. 22–35, June/July 1988.

JORGE, E. et al. A framework for context-aware systems in mobile devices. In: MURGANTE, B. et al. (Ed.). *Computational Science and Its Applications – ICCSA 2012*. [S.l.]: Springer Berlin / Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7336). p. 444–456.

KINDLE. Amazon kindle. 2012. Acesso em: 28 jun. 2012. Disponível em: <http://www.kindle.amazon.com/>.

LARMAN, C. *Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo.* [S.l.]: Bookman, 2005.

LOUREIRO, A. A. F. et al. Computação ubíqua ciente de contexto: Desafios e tendências. *27º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, p. 99–149, 2009.

REMINDERS. Apple ios5. 2012. Acesso em: 25 mai. 2012. Disponível em: <http://www.apple.com/br/ios/features.html>.

RYAN, N.; PASCOE, J.; MORSE, D. Enhanced reality fieldwork: the context-aware archaeological assistant. *Computer Applications in Archeology. Oxford:Tempus*, 1997.

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

SILVA, R. *Suporte ao desenvolvimento e uso de frameworks e componentes.* Dissertação (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson, 2011.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 3. ed. São Paulo: Pearson, 2010.

TIOBE. Tiobe software. 2012. Acesso em: 05 mar. 2012. Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.

ANEXO A – TUTORIAL CONFIGURAÇÃO CAMOBILE

Este documento tem objetivo de apresentar um passo a passo de como configurar o CAMobile para ser utilizado em um projeto.

Para utilizar o CAMobile é necessário adicioná-lo no projeto que será desenvolvido o aplicativo. Também deve se adicionar o *framework Core Location Framework* no projeto e configurar para o aplicativo execute código em *background*, caso o aplicativo necessite verificar um contexto de uso quando não estiver ativo. Feito isso, é necessário importar o *framework* na view que for utilizá-lo, declarar um objeto da classe ContextManager nesta classe e por último chamar o método que inicia o ContextManager no método "viewDidLoad".

Após feito isto, o desenvolvedor pode chamar os métodos que adicionam um Evento para ser executado, após a existência de um determinado Contexto.

Abaixo é mostrado um passo a passo da configuração do CAMobile.

1 - Adicionar *Framework* CAMobile e *Core Location Framework* no projeto.

- Clica no arquivo xcodeproj. - Vai para aba *Build Phases*. - Expande a opção *Link Binary With Libraries*. - Clica em Add itens e adiciona os *frameworks*.

2 - Configurar o projeto para executar em *background*.

- Expande a pasta Supporting Files. - Clica no arquivo [Projeto]-Info.plist. - Clica com o botão direito do mouse e seleciona a opção *Add Row*. - Define a *Key* como "*Required background modes*". - No item 0 do *array* define o *value* como "*App registers for location updates*".

3 - Importar o *framework* na *view* que utiliza o CAMobile.

- #import <CAMobile/CAMobile.h>

4 - Declarar o objeto da classe ContextManager.

- ContextManager *contextManager;

5 - Iniciar o ContextManager.

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4     // Do any additional setup after loading the view, typically
          from a nib.
5
6     // Inicia-se o ContextManager e em seguida os controladores
7     contextManager = [[ContextManager alloc] init];
8     [contextManager startContextManagerForView:self];
9 }
```

Algoritmo A.1: Código Fonte da chamada do método que inicia o ContextManager.