UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

ANDRÉ SALES FONTELES

UM FRAMEWORK PARA AQUISIÇÃO ADAPTATIVA E FRACAMENTE ACOPLADA DE INFORMAÇÃO CONTEXTUAL PARA DISPOSITIVOS MÓVEIS

FORTALEZA, CEARÁ

2013

ANDRÉ SALES FONTELES

UM FRAMEWORK PARA AQUISIÇÃO ADAPTATIVA E FRACAMENTE ACOPLADA DE INFORMAÇÃO CONTEXTUAL PARA DISPOSITIVOS MÓVEIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Profa. Rossana Maria de Castro Andrade, PhD

Co-Orientador: Prof. Windson Viana de Carvalho, DSc.

FORTALEZA, CEARÁ

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

F762f Fonteles, André Sales.
Um framework para aquisição adaptativa e fracamente acoplada de informação contextual para dispositivos móveis. / André Sales Fonteles.
97f. : il. , color., ; 30 cm.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós Graduação em Ciência da Computação, Fortaleza, 2013.
Área de Concentração: Engenharia de Software.
Orientação: Profa. Rossana Maria de Castro Andrade.

1. Computação móvel. 2. Framework (Programa de computador). 3. Engenharia de software. I. Título.

CDD 005

ANDRÉ SALES FONTELES

UM FRAMEWORK PARA AQUISIÇÃO ADAPTATIVA E FRACAMENTE ACOPLADA DE INFORMAÇÃO CONTEXTUAL PARA DISPOSITIVOS MÓVEIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação

Aprovada em: __/__/____

BANCA EXAMINADORA

Profa. Rossana Maria de Castro Andrade, PhD
Universidade Federal do Ceará - UFC
Orientador

Prof. Windson Viana de Carvalho, DSc.
Universidade Federal do Ceará - UFC
Co-orientador

Prof. Fernando Antônio Mota Trinta, DSc.
Universidade Federal do Ceará - UFC

Prof. Jérôme Gensel, DSc.
Université Pierre Mendès France – UPMF

Ao meu Deus, a meus pais, Edenildo
e Suzani, e a minha esposa linda, Lya

AGRADECIMENTOS

Agradeço a Jesus Cristo, por ter me acompanhado como amigo fiel até aqui e pela certeza de que continuará me acompanhando. Por ter me aberto as portas e por ter me apresentado as pessoas que me acompanharam durante essa fase da minha vida. Por ter me dado capacidade e perseverança para escrever esse trabalho. Por ter me dado paz e vida. A ele devo TUDO o que tenho e que sou.

Aos meus orientadores, Professor Windson e Professora Rossana, por terem me guiado e me acompanhado nessa árdua tarefa de concluir um mestrado. Quero sempre me recordar do exemplo e dedicação de Windson ao entrar madrugadas juntamente com seus alunos no laboratório para concluir artigos. Da mesma forma da professora Rossana, que mesmo tão atarefada disponibilizou de seu precioso tempo para me auxiliar, corrigir e orientar.

Aos professores Fernando Trinta e Jérôme Gensel, que compõem a banca examinadora e certamente contribuirão com o bom resultado desse trabalho.

Também sou grato a meus pais, Edenildo e Suzani, pelo apoio, carinho e pelo exemplo que eles sempre me deram.

Sou grato de igual forma a minha esposa Lya, que me apoiou e auxiliou desde o início. Pela sua cumplicidade, seu amor e pela sua paciência e carinho quando estive cansado.

Não poderia também deixar de agradecer ao meu amigo Benedito Neto, por ter me ajudado em todas as minhas empreitadas durante todo o mestrado e por também ter sido um exemplo de caráter.

Ao Professor Carlos André, que me auxiliou na complexidade dos algoritmos.

Ao amigo e gerente Bruno Sabóia, que sempre flexibilizou meus horários para que eu resolvesse as coisas do mestrado.

Também agradeço a todos os amigos e colegas do GREat que me acompanharam e compartilharam experiências durante o mestrado. Em especial ao Ricardo(Zezim), Nayane, Rafael Lima, Ismayle, Rainara, Charles, Adyson.

Enfim, a todos que me ajudaram de alguma forma na conclusão desse trabalho.

“Se, com a tua boca, confessares Jesus como Senhor e, em teu coração, creres que Deus o ressuscitou dentre os mortos, serás salvo.”

(Romanos 10:9)

RESUMO

Dispositivos móveis, tais como *smartphones* e *tablets*, dotados de uma série de sensores se tornaram comuns no nosso dia a dia. Esse cenário propiciou que aplicações dessas plataformas acessassem cada vez mais informações contextuais do ambiente, do sistema e do usuário para se adaptar de acordo ou oferecer serviços relevantes. Aplicações dotadas desse comportamento são conhecidas como sensíveis ao contexto. Várias infraestruturas já foram criadas para auxiliar no desenvolvimento de aplicações desse tipo. Essas infraestruturas facilitam a aquisição e o gerenciamento de informações contextuais. Todavia, muitas delas não são apropriadas para o ambiente de execução heterogêneo e peculiar dos dispositivos móveis. Esse trabalho de dissertação de mestrado apresenta uma infraestrutura para aquisição de contexto chamada CAM (Context Acquisition Manager). CAM é um *framework* projetado para utilização em dispositivos móveis dotados de sensores embarcados. Entre suas principais características estão o fraco acoplamento entre ele e as aplicações que o utilizam e a possibilidade de adaptação no momento de implantação ou de execução. A adaptação na implantação permite ao desenvolvedor personalizar quais características serão incluídas na instalação do *framework*. Já a adaptação em tempo de execução permite desabilitar ou habilitar partes do *framework* conforme a demanda. Para avaliação desse trabalho foi desenvolvida uma aplicação sensível ao contexto como prova de conceito que utiliza o *framework* CAM. Através do desenvolvimento dela, foi possível perceber a clara separação entre o código de aquisição de contexto, encapsulado no *framework*, do código de uma aplicação que o utiliza. Também foi desenvolvido um protótipo de uma outra aplicação, no qual foram realizados testes do mecanismo de adaptação dinâmica do *framework*. No experimento foi analisado o impacto da adaptação na utilização do processador e da memória primária do dispositivo, que mostrou um aumento na economia de ambos.

Palavras-chave: Sensibilidade ao Contexto. Adaptação de Software. Dispositivos Móveis.

ABSTRACT

Mobile devices, such as smartphones and tablets, with a number of sensors have become commonplace in our daily lives. This scenario promotes applications from these platforms to increasingly access contextual information of the environment, the user and the system, which adapt accordingly or offer relevant services. This behavior is known as context-awareness. Several infrastructures have been created to help in the development of context-aware applications. These infrastructures facilitate the acquisition and management of contextual information. However, many of them are not appropriated to the heterogeneous and particular environment of mobile devices. This work presents an infrastructure for context acquisition called CAM (Context Acquisition Manager). CAM is a framework designed for use in sensor rich mobile devices. Among its main features are the loosely coupling with the applications that use it and the possibility of adapting in deployment time or execution time. The deployment adaptation allows a developer to customize what features will be included in the installation of the framework. The adaptation in execution time allow the framework to enable or disable its features according to applications requirements. To evaluate this work we developed a context-sensitive application as a proof of concept that uses the framework CAM. Through the development of this application, it was possible to notice a clear separation between the context acquisition code, wrapped by the framework, and the application code. A prototype of another application in which tests were performed on the dynamic adaptation mechanism of the framework was also developed. In this experiment, the impact of the adaptation on the resources of the device was investigated, which showed an increased economy in memory and CPU.

Keywords: Context Awareness. Software Adaptation. Mobile Devices.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Dispositivos móveis tais como celulares, *tablets* e computadores de mão, se tornaram populares e comuns em nosso dia a dia. Acessar, a partir destes dispositivos, aplicativos de redes sociais, e-mails, bate-papos virtuais, a qualquer hora e em qualquer lugar, faz parte do cotidiano de muitas pessoas. Este é o caminho para a concretização do conceito de Computação Ubíqua predito por Weiser (1991). Para ele, a Computação Ubíqua teria como característica marcante a invisibilidade da tecnologia na vida das pessoas. Dessa forma, os usuários passariam a interagir com computadores tão naturalmente que esse fato passaria a ser cotidiano e natural.

Atingir uma interação natural entre usuários e dispositivos é uma tarefa complexa, pois aplicações desenvolvidas para dispositivos móveis estão inseridas em um ambiente altamente dinâmico devido à mobilidade do usuário. Por um lado, a mobilidade permite que o acesso à localização do usuário seja utilizado para fornecer serviços altamente relevantes, mas, por outro lado, pode causar problemas como a perda de sinal de telefonia ou o esgotamento da fonte de energia (bateria). Essa dinamicidade exige que um software de um dispositivo móvel esteja em contínua adaptação ao ambiente para obter sucesso (PREUVENEERS; BERBERS, 2007), característica conhecida como sensibilidade ao contexto.

Várias infraestruturas, tais como *frameworks*[1], *middlewares*[2], já foram desenvolvidas para auxiliar a criação de sistemas sensíveis ao contexto, como pode ser visto em (BALDAUF et al., 2007) e (HONG et al., 2009). Essas infraestruturas facilitam que aplicações obtenham as informações sobre o ambiente e se adaptem de acordo. Por exemplo, uma aplicação pode utilizar uma infraestrutura que adquira informações contextuais e dispare um evento quando um estado contextual (e.g., temperatura > 20 e dia = ensolarado) determinado pela aplicação for alcançado. A aplicação ao ser notificada pela infraestrutura poderia então adaptar seu comportamento de acordo com o contexto.

[1] Segundo Johnson (1997), um *framework* pode ser considerado como um projeto reutilizável de um sistema, ou parte dele, e que é representado como um conjunto de classes abstratas e da forma como elas interagem.

[2] Para (BRUNEO et al., 2007), *middleware* é uma camada de software entre sistema operacional e aplicação que oferece um alto grau de abstração em computação distribuída.

1.2 Motivação

Infraestruturas de suporte ao desenvolvimento de aplicações sensíveis ao contexto provêem mecanismos eficazes para iniciar e, até mesmo, adaptar as aplicações. Entretanto, elas mesmas, em geral, não são adaptáveis (KELING et al., 2012). Tal característica trás dificuldades na utilização dessas infraestruturas em dispositivos móveis (DM). Preuveneers e Berbers (2007) destacam três dessas dificuldades:

1. Muitas infraestruturas de gerenciamento de contexto não suportam nativamente uma instalação personalizada e adaptada de seus recursos para um determinado DM (implantação com política de tudo-ou-nada, isto é, ou se instala todos os módulos e componentes da infraestrutura ou não se utiliza ela).

2. A quantidade mínima de recursos requeridos pelas infraestruturas está além da que um DM pode oferecer.

3. Muitas infraestruturas não são construídas levando em consideração a autonomia limitada que uma bateria de DM oferece.

A politica do tudo-ou-nada, adotada por muitas dessas infraestruturas, impede uma possível economia de recursos decorrente de uma instalação personalizada e pode, até mesmo, impedir a utilização da infraestrutura em determinados cenários. Por exemplo, caso uma infraestrutura possua partes referentes à utilização de GPS e acelerômetro, mesmo que o celular não disponha desses sensores, é comum que essas partes estejam incluídas na implantação e que utilizem recursos do dispositivo. Isso pode acarretar em um desperdício de recursos (classes serão instanciadas, mas não serão utilizadas) ou erros de implantação impossibilitarão o processo de ativação da aplicação (e.g., ausência de API de GPS). A alta heterogeneidade de modelos de DM requer que uma instalação possa ser adaptável. Mesmo em uma única plataforma, como por exemplo Android, existem diferenças que variam de sensores disponíveis até a própria versão do sistema operacional.

Outra característica das aplicações móveis e sensíveis ao contexto que pode afetar o desempenho das infraestruturas de suporte é o fato de que essas aplicações podem ter interesses passageiros em determinados elementos contextuais (e.g., localização do usuário, temperatura). Por exemplo, uma aplicação pode ter interesse em saber a localização do usuário apenas uma

vez por dia. Nesse caso, com o passar do tempo, trechos de código responsáveis por sensorear ou observar determinados elementos contextuais que não são mais de interesse das aplicações poderiam ser removidos ou desligados para economizar recursos. Da mesma forma, caso um novo elemento se tornasse de interesse, um componente responsável por observá-lo poderia ser adicionado ou ligado, se houvesse suporte do hardware. Assim, tais infraestruturas poderiam dar suporte à adaptação de suas próprias configurações internas em tempo de execução para alcançar um melhor gerenciamento de recursos e requisitos, todavia a maioria delas não dispõe de tal propriedade (KELING et al., 2012).

Apesar das dificuldades expostas por Preuveneers e Berbers (2007) na utilização dessas infraestruturas, aplicações sensíveis ao contexto que não fazem uso delas costumam sofrer de um alto acoplamento entre o código responsável pela aquisição de contexto e o restante da aplicação (BALDAUF et al., 2007). Esse acoplamento dificulta a prática de reúso de código e torna mais complexo o acesso às informações contextuais pela falta de uma interface padronizada para tal.

1.3 Objetivos e Contribuições

Tendo em vista as dificuldades apresentadas na utilização de infraestruturas de suporte a criação de sistemas sensíveis ao contexto, esse trabalho propõe um *framework* de aquisição de contexto adaptável chamado CAM (Context Acquisition Manager). O *framework* aqui proposto busca trazer a adaptação, necessária aos sistemas móveis, para a própria infraestrutura de gerenciamento de contexto, mais especificamente para a aquisição de contexto. Essa adaptação ocorre por meio da adição, remoção, inicialização ou parada de componentes do *framework*.

O *framework* CAM é projetado para utilização embarcada em um único DM e, além de realizar aquisição de contexto, permite:

1. implantação personalizada para as características próprias de cada dispositivo; e
2. adaptação em tempo de execução aos requisitos das aplicações que a utilizam.

A implantação personalizada diminui problemas causados pela heterogeneidade dos DM e possibilita uma economia dos recursos utilizados, uma vez que partes desnecessárias do

framework não estarão instaladas e nem executando. A adaptação em tempo de execução, ou adaptação dinâmica, também traz como benefício o melhor gerenciamento de recursos do DM. Além disso, facilita o gerenciamento a mudanças de requisitos das aplicações que utilizam o *framework* proposto, pois componentes que satisfaçam requisitos podem ser adicionados ou removidos em tempo de execução.

Para que tais níveis de adaptação sejam alcançados, o *framework* proposto utiliza uma arquitetura orientada a componentes, como pode ser visto na Figura 1.1. A arquitetura orientada a componentes permite que partes (componentes) do sistema sejam facilmente trocadas, adicionadas ou removidas. Os principais elementos presentes nessa arquitetura são componentes de aquisição de contexto (`CAC`), uma infraestrutura para gerenciamento do ciclo de vida dos `CACs` (`CAC Manager`) e um componente responsável por analisar e planejar adaptações em tempo de execução, chamado de `Adaptation Reasoner`. A aplicação, é quem utiliza o *framework* e o contexto adquirido.

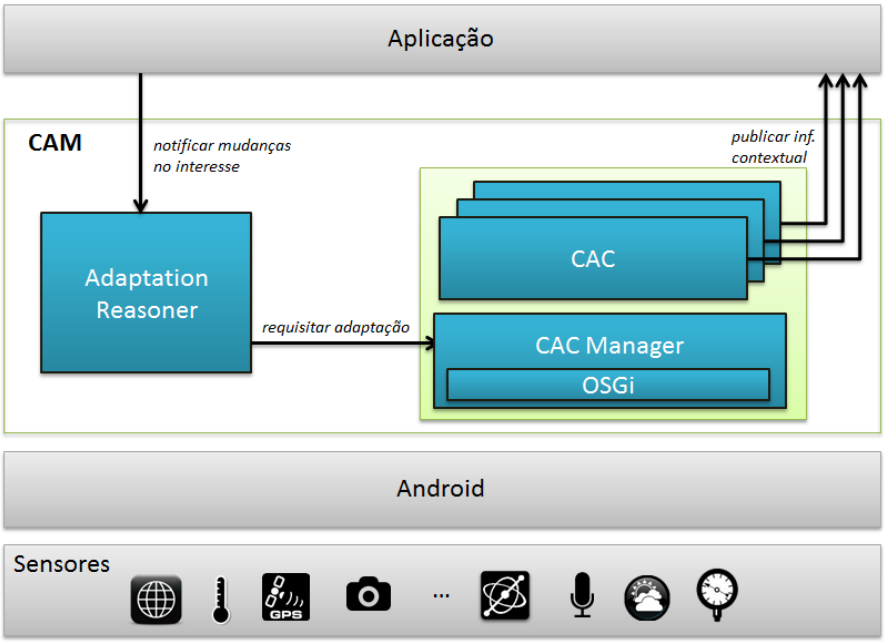

Figura 1.1: O framework CAM implantado em um dispositivo Android.

Os `CAC` são as unidades básicas de composição do *framework*. Eles são responsáveis por adquirir informações contextuais através de APIs do sistema operacional (Android) de acesso a sensores embarcados do DM (e.g., acelerômetro e GPS) ou a partir de inferências e agregação entre informações contextuais de mais baixo nível. Um exemplo de inferência seria unir as informações do acelerômetro e de localização ao longo do tempo para descobrir que o usuário está correndo. Um `CAC` também pode adquirir informações contextuais a partir de

sistemas externos, como serviços de meteorologia ou trânsito. Todo `CAC` executa sobre o gerenciamento do `CAC Manager`, que oferece suporte à instalação, remoção, execução e parada dos mesmos através do uso do OSGi, que será explicado na Seção 2.4.2.

Um dos pontos centrais da construção do *framework* é a concepção de um algoritmo capaz de gerar configurações estruturais válidas dos componentes de aquisição de contexto. Esse algoritmo está presente no `Adaptation Reasoner` e deve levar em consideração os requisitos das aplicações que utilizam o *framework* bem como possíveis dependências entre um ou mais `CACs`.

1.4 Metodologia

A metodologia científica utilizada neste trabalho é caracterizada resumidamente nos itens a seguir.

1. **Revisão de Literatura**: Inicialmente é realizada uma revisão bibliográfica sobre o conceito de Sensibilidade ao Contexto, onde são apresentadas as principais técnicas e infraestruturas de suporte ao desenvolvimento desse tipo de sistema. Também é realizada uma revisão bibliográfica sobre a adaptação de *software* com foco na computação Móvel. Nesse ponto são levantadas os principais tipos de adaptação e técnicas de implementação das mesmas. Por fim, é realizado um estudo sobre a Engenharia de Software Baseada em Componentes, paradigma que auxilia na implementação de sistemas capazes de se adaptar em tempo de projeto, implantação e execução.

2. **Revisão de Trabalhos Relacionados**: Além da revisão inicial sobre as principais infraestruturas de suporte ao desenvolvimento de sistemas sensíveis ao contexto, também são levantadas aquelas que mais se assemelham ao *framework* proposto. O critério utilizado para tal seleção é se a infraestrutura em questão é desenvolvida para utilização embarcada em dispositivo móvel ou se ela apresenta algum tipo de adaptação.

3. **Definição do Framework proposto**: O passo seguinte deste trabalho de dissertação consiste em definir um *framework* para aquisição de contexto adaptável. Para sua definição, são utilizados conceitos de adaptação de *software* e de Engenharia de Software Baseada em Componentes, apresentadas anteriormente.

4. **Avaliação dos Resultados**: Após definido o *framework* proposto, é apresentada a implementação passo a passo de uma aplicação prova de conceito e analisados os principais benefícios alcançados pela utilização do *framework*. Também é implementado um protótipo de uma segunda aplicação para a realização de testes sobre o impacto da utilização do *framework* proposto e sua adaptação sobre o uso de memória e CPU do dispositivo móvel.

1.5 Estrutura do Documento

Esta dissertação está organizada em seis capítulos. O presente capítulo descreve uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, os objetivos e as contribuições deste trabalho.

No Capítulo 2 são abordados os principais conceitos teóricos necessários para compreensão deste trabalho. Entre eles estão sensibilidade ao contexto, definição de contexto adotada e técnicas de suporte a criação de aplicações sensíveis ao contexto. Também são apresentadas a adaptação de *software* na computação móvel e a Engenharia de Software Baseada em Componentes.

No Capítulo 3 são apresentados importantes infraestruturas de aquisição de contexto encontradas na literatura. Em seguida, são apresentadas como trabalhos relacionados outras infraestruturas mais semelhantes ao *framework* proposto. Para a identificação delas, foi analisado se essas infraestruturas entravam em pelo menos um dos dois critérios a seguir: elas são projetadas para dispositivos móveis ou realizam algum tipo de adaptação.

O Capítulo 4 apresenta o *framework* proposto, seu modelo de contexto e de componentes. Também é apresentado o *middleware* LoCCAM, que faz uso do *framework* proposto, e as principais contribuições realizadas na integração entre CAM e LOCCAM.

No Capítulo 5 são apresentados, através da implementação de uma prova de conceito, exemplos de código de utilização do *framework* proposto e do LoCCAM. Nesse capítulo também são apresentados os resultados obtidos a partir de testes realizados no mecanismo de adaptação dinâmica.

Por fim, o Capítulo 6 descreve de forma sucinta os resultados alcançados, bem como as conclusões deste trabalho e publicações decorrentes. Além disso são apresentadas possíveis

melhorias a serem consideradas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos teóricos necessários para compreensão deste trabalho. A Seção 2.1 apresenta a definição de conceito adotada nessa dissertação. Já a Seção 2.2 apresenta Sensibilidade ao Contexto e discute técnicas de suporte ao desenvolvimento de sistemas sensíveis ao contexto. Na Seção 2.3, o tema de adaptação de software na computação móvel é abordado. Por fim, a Engenharia de Software Baseada em Componentes é discutida na Seção 2.4.

2.1 Definição de Contexto

De acordo com Dey e Abowd (1999), sistemas sensíveis ao contexto devem ser capazes de prover informações e serviços relevantes ao usuário dependendo de que tarefas estes estejam executando. Esses sistemas são capazes de adaptar suas operações de acordo com o contexto corrente sem a intervenção explícita do usuário, aumentando assim a usabilidade e a efetividade do sistema (BALDAUF et al., 2007). Para Dey e Abowd (1999) contexto é definido como toda e qualquer informação que possa ser usada para caracterizar uma entidade. Sendo que uma entidade é uma pessoa, local ou objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo os próprios usuários e a aplicação.

Apesar da definição de contexto de Dey e Abowd (1999) ser frequentemente referenciada (BETTINI et al., 2010), no desenvolvimento deste trabalho foi adotada uma definição mais focada na aquisição de contexto expressa por Viana (2010). Segundo essa definição, os elementos que compõem o contexto (e.g., temperatura ambiente e localização do usuário) são definidos baseados na relevância que possuem para o sistema e na possibilidade que o sistema possui de sensoreá-los em um determinado instante do tempo.

Nessa definição, contexto é definido como a interseção entre dois conjuntos dinâmicos e evolutivos, como mostra a Figura 2.1. Um conjunto é chamado de Zona de Interesse (ZI) e é constituído de todas as informações do ambiente, do sistema e do usuário que o sistema gostaria de conhecer. O outro conjunto é denominado Zona de Observação (ZO), onde estão presentes todas as informações do ambiente, do sistema e do usuário que o sistema é capaz de obter. A interseção entre o que é de interesse do sistema e o que o sistema consegue obser-

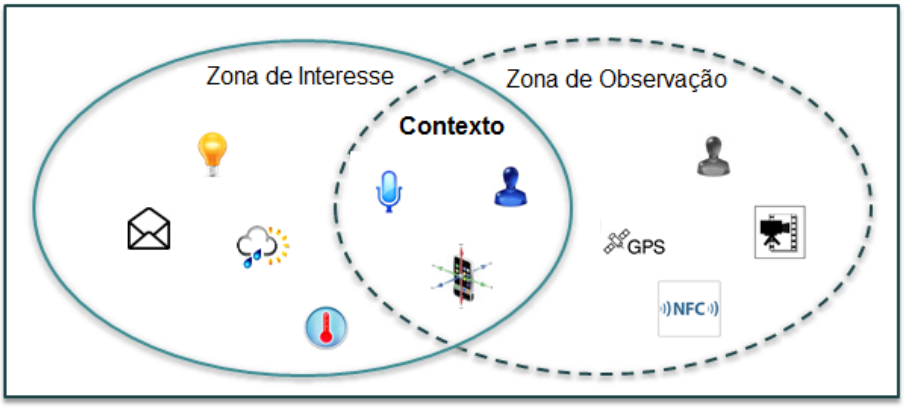

Figura 2.1: Contexto como interseção entre Zona de Interesse e Zona de Observação. Adaptado de Viana (2010).

var (ZI ∩ ZO) em um instante t é considerada contexto. Por exemplo, suponha um programa para celular que, de acordo com o som ambiente, ajuste o nível do alerta sonoro da campainha. Nesse caso, o som ambiente é um elemento da ZI, pois o sistema almeja essa informação. Através do microfone do dispositivo, é possível capturar o som ambiente, logo ele também está na ZO. Finalmente, por estar nos dois conjuntos, o som ambiente é considerado contexto pela aplicação.

Os elementos presentes na ZI e na ZO podem mudar com o tempo. Ainda no exemplo anterior, com o passar do tempo, o usuário pode configurar o seu celular para o modo silencioso. Caso isso ocorra, seu dispositivo não emitirá mais alertas sonoros de forma alguma. Logo, já não importa para o sistema conhecer o som ambiente para ajustar o alerta. Isso faz com que o som ambiente já não esteja presente na ZI, apesar de ainda estar na ZO. Por não fazer parte dos dois conjuntos, o som ambiente não seria mais considerado contexto. Da mesma forma, ainda que o dispositivo não estivesse no modo silencioso, seu microfone poderia estar defeituoso, isso impossibilitaria que o som ambiente fosse capturado. Portanto, ele já não estaria na ZO e não faria mais parte do contexto, apesar de ainda fazer parte da ZI. O comportamento descrito acima é dito como caráter evolutivo do contexto. Assim, podemos dizer que os elementos observados que compõem o contexto no tempo t não são necessariamente os mesmos em outro instante t', ou seja, não se pode afirmar que $Contexto(t) = Contexto(t')$.

2.2 Sensibilidade ao Contexto

O funcionamento de sistemas sensíveis ao contexto possui como fluxo básico a execução das seguintes tarefas:

1. adquirir informações contextuais de baixo nível;
2. elevar o nível dessas informações por meio de técnicas, como inferências ou transformações, para que elas se tornem utilizáveis pela aplicação;
3. armazenar as informações para que elas possam ser acessadas posteriormente e/ou gerar eventos quando determinados estados de contexto são detectados; e
4. oferecer serviços ou informações de acordo com o contexto.

Dentro das definições citadas anteriormente, é possível encontrar uma variedade de sistemas sensíveis ao contexto. Por exemplo, a aplicação GREat Tour (MARINHO et al., 2012) que é um guia de visitas de um laboratório de pesquisa da Universidade Federal do Ceará. Essa aplicação executa em dispositivos móveis de visitantes e provê informação sobre o ambiente e pesquisadores de cada laboratório em que o usuário está visitando. A aplicação fornece informações e funcionalidades diferentes de acordo com dados contextuais de localização, perfil e preferências do usuário e características do dispositivo. Outros aplicativos sensíveis ao contexto, como PhotoMap (VIANA et al., 2007) e Captain (BRAGA et al., 2012), utilizam informações contextuais para enriquecer dados multimídia.

Neste trabalho foi considerado como sensibilidade ao contexto a capacidade de prover informações e serviços relevantes dependendo não apenas da tarefa que o usuário está executando, mas dos contextos do sistema, ambiente e do próprio usuário.

2.2.1 Desenvolvimento de Sistemas Sensíveis ao Contexto

Para facilitar o desenvolvimento, a extensibilidade e reusabilidade em sistemas sensíveis ao contexto, várias infraestruturas de suporte foram propostas e desenvolvidas, como poderá ser visto na Seção 3.1. Dey et al. (2001) elicitaram seis requisitos comuns a aplicações sensíveis ao contexto que podem ser suportados por tais infraestruturas:

1. **Separação de Interesses** - Consiste na separação de como o contexto é adquirido do código que o utiliza. Este requisito permite que os desenvolvedores de aplicações concentrem seus esforços em como as aplicações devem reagir a mudanças de contexto, deixando os detalhes de interações com sensores para os desenvolvedores das infraestruturas.

2. **Interpretação de Contexto** - Uma aplicação pode não estar interessada em informações de baixo nível fornecidas diretamente por sensores. As infraestruturas de suporte podem, portanto, oferecer serviços de enriquecimento e interpretação de informações contextuais para níveis semânticos mais elevados. Por exemplo, uma aplicação pode estar interessada em saber o endereço onde o usuário se encontra, porém o GPS fornece a localização em latitude e longitude. Nesse caso, um serviço que transforme latitude e longitude em endereço pode ser disponibilizado pela infraestrutura.

3. **Comunicação Transparente e Distribuída** - É possível que sensores fornecedores de informações contextuais não estejam embarcados no mesmo computador ou dispositivo móvel, no qual as aplicações sensíveis ao contexto estão sendo executadas. Nesse caso, a interação entre sensores e aplicações devem ocorrer de forma transparente, sem preocupações com detalhes de rede.

4. **Constante Disponibilidade de Aquisição de Contexto** - Informações contextuais devem estar sempre disponíveis quando as aplicações requisitarem. Esse requisito é parcialmente viável, dado as características dinâmicas e voláteis de ambientes ubíquos nos quais aplicações sensíveis ao contexto estão normalmente inseridas.

5. **Armazenamento de Contexto** - Aplicações sensíveis ao contexto podem requerer um histórico das informações contextuais para tomar decisões. Por exemplo, uma aplicação pode desejar saber se é a primeira vez que um usuário encontra-se em um local para apresentar uma mensagem de boas vindas. Dessa foram, um requisito comum a essas aplicações é um armazenamento de contextos anteriores ao atual.

6. **Descoberta de Recursos** - Para que aplicações possam requisitar informações contextuais de uma infraestrutura, é necessário que ela conheça quais informações estão disponíveis. Por exemplo, um mecanismo de Descoberta de Recursos pode permitir que

aplicações requisitem recursos e recebam uma referência para componentes que os ofereçam.

Alguns exemplos de infraestruturas criadas que suportam todos ou alguns desses requisitos são: Context Toolkit (DEY et al., 2001), CASS (FAHY; CLARKE, 2004), SOCAM (GU et al., 2005), MobiCon (LEE et al., 2012). Apesar de peculiaridades presentes em cada infraestrutura, Baldauf et al. (2007) identificaram o que eles chamaram de uma "arquitetura conceitual"recorrente na maioria delas, como visto na Figura 2.2. Nessa arquitetura são definidas as principais camadas de tais infraestruturas e suas funcionalidades.

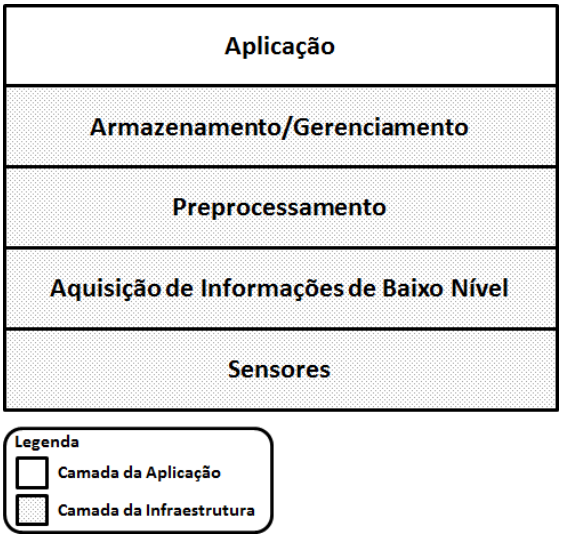

Figura 2.2: Camadas recorrentes em infraestruturas de suporte a aquisição de contexto. Adaptado de (BALDAUF et al., 2007).

A primeira camada, a mais inferior, é uma coleção de diferentes sensores. Tais sensores podem ser físicos, lógicos ou virtuais (INDULSKA; SUTTON, 2003). Sensores físicos são sensores com *hardware* reais (tais como GPS, acelerômetro e termômetro). Sensores lógicos oferecem informações contextuais a partir de *softwares* ou serviços (e.g., calendários virtuais e *webservices* de informações climáticas), por fim, sensores lógicos oferecem informações contextuais obtidas de inferências realizadas a partir de informações de outros sensores. Na arquitetura de muitas infraestruturas, a maioria ou a totalidade destes sensores não se encontram embarcados em dispositivos como *smartphones* e *tablets*, ao invés disso, eles são fisicamente distribuídos em um ou mais ambientes. Tais arquiteturas são projetadas para interagir com ambientes ubíquos ricos em sensores.

A segunda camada é responsável por recuperar as informações de baixo nível a partir da camada de sensores. Tais informações são obtidas por meio de *drivers* apropriados, no caso de sensores físicos, ou de APIs, para sensores virtuais ou lógicos. Essa camada costuma fazer uso de componentes de *software* que encapsulam a complexidade de interação com sensores heterogêneos. Além disso, componentes que forneçam o mesmo tipo de informação contextuais e possuam interface padronizadas podem ser trocados. Por exemplo, um componente que forneça a localização do usuário por meio de GPS pode ser trocado por outro que forneça a mesma informação por meio de RFID, desde que possuam a mesma interface.

A terceira camada, chamada de Preprocessamento, não está presente em todas as infraestruturas e é responsável por gerar informações de mais alto nível a partir das informações obtidas das camadas inferiores por meio de inferências e agregações. Outra funcionalidade presente nesta camada é a de escolher qual informação será utilizada quando existem conflitos. Por exemplo, caso sensores de GPS e RFID forneçam o mesmo tipo de informação de localização do usuário, contudo com valores diferentes.

A camada de Armazenamento/Gerenciamento organiza toda a informação adquirida pelas camadas inferiores e as disponibiliza por meio de uma interface acessível pela camada de Aplicação, que fica logo acima. A camada Aplicação é quem de fato reage oferecendo serviços ou se adaptando de acordo com as informações contextuais providas.

Apesar da arquitetura recorrente observada por Baldauf et al. (2007) ser útil para a compreensão geral do funcionamento interno de sistemas sensíveis ao contexto, a visão apresentada por ela é abstrata para ser utilizada como referência na construção de sistemas reais. Marinho et al. (2012) propõe uma linha de produto de *software* para aplicações sensíveis ao contexto que possui uma arquitetura de referência mais detalhada, como pode ser visto na Figura 2.3. Essa arquitetura de referência adota uma abordagem orientada a serviços e nela estão presentes quatro camadas:

1. **Camada de Sensores de Contexto**: provê uma abstração para a captura de contexto de baixo nível através de sensores para as camadas superiores. Essa camada é equivalente Camada de Aquisição de Informações de Baixo Nível de Baldauf et al. (2007).

2. **Camada de Gerenciamento de Contexto**: é responsável por receber e armazenar os dados de contexto, recuperando-os quando requisitados por camadas superiores. Ela é

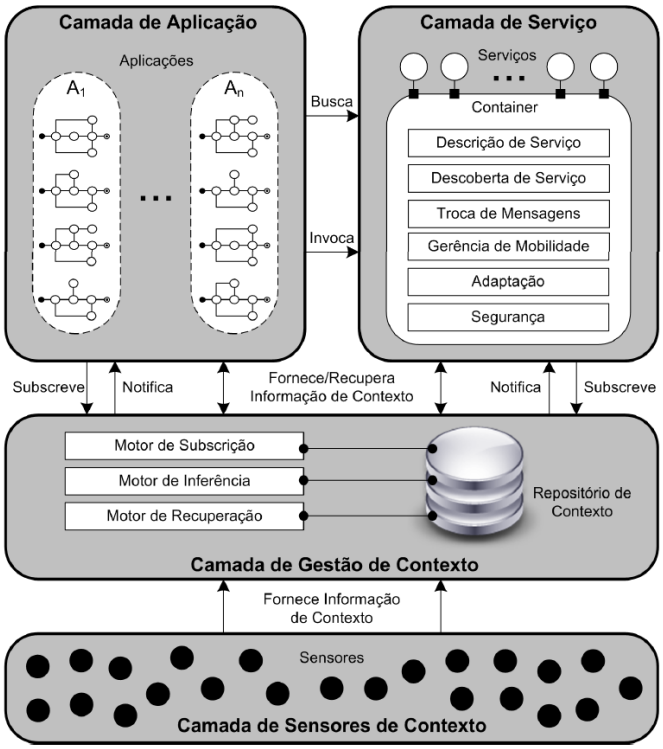

Figura 2.3: Arquitetura de referência para aplicações sensíveis ao contexto (MARINHO et al., 2012).

composta por: Repositório de Contexto, Motor de Recuperação, Motor de Inferência e Motor de Subscrição. O Repositório de Contexto é onde todas as informações são salvas. O Motor de Recuperação permitirá que as informações do repositório sejam acessadas pelas camadas superiores. Para derivar informações contextuais de alto nível é utilizado o Motor de Inferência. Por fim, o Motor de Subscrição é responsável por notificar a Camada de Serviço e Camada de Aplicação, de forma assíncrona quando um determinado contexto predefinido for detectado. Essa camada é equivalente as camadas de Preprocessamento e Armazenamento/Gerenciamento de Baldauf et al. (2007).

3. **Camada de Serviço**: gerencia ciclo de vida de serviços e provê mecanismos de comunicação entre eles mesmos e entre aplicações.

4. **Camada de Aplicação**: é composta de aplicações sensíveis ao contexto que acessam a Camada de Serviço e a Camada de Gerenciamento de Contexto. Essas aplicações são desenvolvidas utilizando serviços da Camada de Serviços. Essa camada, juntamente com a de serviço, é equivalente a Camada de Aplicação de Baldauf et al. (2007).

2.2.2 Modelagem de Contexto

Outro ponto importante na criação de aplicações sensíveis ao contexto se refere a forma como a informação contextual será modelada e representada. Uma boa modelagem de contexto tem a capacidade de reduzir a complexidade de aplicações sensíveis ao contexto além de facilitar a manutenção e evolução das mesmas (BETTINI et al., 2010). Ainda segundo Bettini et al. (2010), os principais requisitos na definição de um modelo de contexto são:

1. **Heterogeneidade e Mobilidade**: Um modelo de contexto deve ser capaz de expressar diversos tipo de informações contextuais. Além disso, muitos sistemas sensíveis ao contexto são móveis ou possuem sensores móveis. Assim, o modelo também deve ser capaz lidar com uma constante mudança de ambiente ao qual as informações estão atreladas.

2. **Relações e Dependências**: Várias relações podem existir entre informações contextuais. Entre elas está a de dependência. Essa relação significa que uma informação pode estar intimamente ligada a outra. Por exemplo, uma informação de quanto tempo de bateria resta em um dispositivo móvel pode variar caso o usuário esteja com o GPS ligado ou não.

3. **Histórico Contextual**: Aplicações sensíveis ao contexto podem necessitar de informações passadas ou futuras (adquiridas por meio de inferências). Assim, outro requisito necessário para um modelo de contexto é lidar com históricos contextuais e formas de expressar o tempo em que determinado contexto ocorreu ou ocorrerá.

4. **Imperfeição**: Devido a dinamicidade, heterogeneidade e inacurácia de sensores, informações contextuais podem ser imprecisas e até mesmo erradas. Além disso, informações contextuais oriundas de diferentes fontes podem ser conflitantes. Um modelo de contexto deve incluir a qualidade de uma informação para que aplicações e motores de inferência possam arrazoar sobre as informações.

5. **Usabilidade da Modelagem Formal**: Essa característica se refere a capacidade com que desenvolvedores podem traduzir conceitos do mundo real ao modelo contextual.

6. **Eficiência de Acesso a Informações**: Informações contextuais relevantes devem ser facilmente acessadas.

Uma questão relevante na concepção de um modelo contextual se refere as abordagens para implementação do modelo. Strang e Linnhoff-Popien (2004) fazem um levantamento das principais técnicas utilizadas para tanto:

1. **Chave-Valor**: essa é a forma mais simples de implementar modelagem de contexto. Nela, informações são representadas como um conjunto de atributos (chave) e respectivo valor (e.g., temperatura=30). Sua principal vantagem é a simplicidade, todavia essa abordagem possui dificuldade em expressar detalhes de informações contextuais e é pouco formal.

2. **Linguagens de Marcação**: abordagem que expressa a modelagem de contexto por meio linguagens de marcação como o XML. Esse tipo permite representações mais robustas que a chave-valor devido à estrutura hierárquica dessas linguagens e alguma validação quando usada em conjunto com restrições de marcação, como XML Schema e DTD.

3. **Modelos Gráficos**: essa técnica de modelagem consiste em representar graficamente o modelo de contexto, por exemplo, utilizando UML. É uma abordagem com maior foco nos humanos. A partir da modelagem gráfica é possível derivar algum código.

4. **Modelos Orientados a Objetos**: essa abordagem obtém benefícios do paradigma da Orientação a Objetos como herança, encapsulamento e reusabilidade. É possível realizar uma validação parcial por meio, por exemplo, de compiladores.

5. **Modelos Baseados em Lógica**: nessa abordagem o modelo de contexto é definido como fatos, expressões lógicas e regras. Sua principal característica é o alto grau de formalismo, que permite a realização de inferências, porém também apresenta um maior grau de complexidade na tradução do mundo real para o modelo de contexto.

6. **Modelos Baseados em Ontologias**: possuem uma estrutura adequada para a representação do mundo real, seus objetos e relacionamentos e é facilmente compreensível por computadores.

2.3 Adaptação de Software na Computação Móvel

2.3.1 Definição

Adaptação de software é um termo amplo, extensamente estudado em múltiplas disciplinas e usado para denotar qualquer tipo de modificação em qualquer momento do ciclo

de vida de um sistema (KAKOUSIS et al., 2010). Espera-se que através da adaptação, um software seja capaz de satisfazer seus requisitos, em resposta a mudanças do sistema e do seu contexto até mesmo em tempo de execução (SALEHIE; TAHVILDARI, 2011). A adaptação em tempo de execução é uma característica chave para a computação móvel e sensível ao contexto, pois proporciona uma interação mais natural entre usuário e sistema e possivelmente sem necessidade de intervenção explícita por parte do usuário.

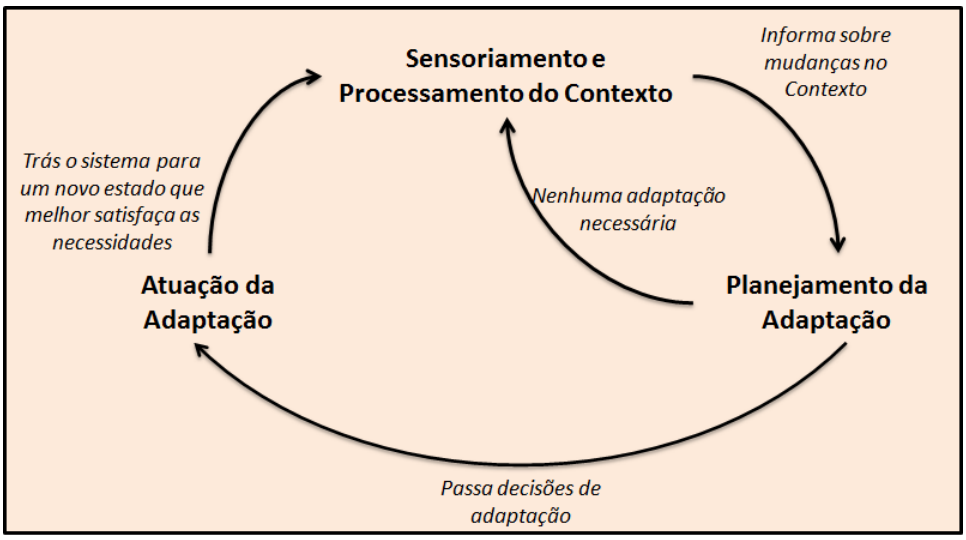

Figura 2.4: Processo de adaptação na computação ubíqua. Adaptado de (KAKOUSIS et al., 2010).

Kakousis et al. (2010) definem a adaptação na computação ubíqua, termo amplo que pode abranger a computação móvel, como sendo o processo de reação disparado por um evento ou um conjunto de eventos em determinado contexto de execução. Senso assim, é importante que aplicações móveis sejam capazes de sensorear o contexto em que estão inseridas para reagir de acordo, se necessário, quando houver mudanças no mesmo. Ainda segundo os autores Kakousis et al. (2010), esse processo de reação descrito pode ser dividido em três fases: Sensoriamento e Processamento do Contexto (*Context Sensing and Processing*), Análise e Planejamento da Adaptação (*Adaptation Reasoning and Planning*) e Atuação da Adaptação (*Adaptation Acting*). A Figura 2.4 apresenta o processo da adaptação ubíqua e suas fases. A fase de Sensoriamento e Processamento do Contexto consiste no processo de aquisição e enriquecimento do contexto de uma aplicação. Quando um novo contexto é detectado, ocorre a fase de Análise e Planejamento da Adaptação. Nela, o sistema é chamado a analisar o novo contexto e decidir o que necessita ser alterado e como isso deve ocorrer para que os objetivos globais da adaptação sejam alcançados. A fase de Atuação da Adaptação é aquela em que o sistema

utiliza os mecanismos de adaptação apropriados para implementar as decisões tomadas na fase de análise do novo contexto.

2.3.2 Taxonomia

Existem diversas formas de se categorizar o processo de adaptação. Por exemplo, os autores Ketfi et al. (2002) propõem uma classificação em quatro tipos baseada nos **objetivos gerais** da adaptação:

- A **Adaptação Corretiva** (*Corrective Adaptation*) ocorre quando a aplicação não está se comportando de forma correta. Essa adaptação busca corrigir um comportamento errado de uma aplicação. Por exemplo, caso um componente não esteja funcionando, ele pode ser substituído por outro que realize exatamente a mesma função, porém com funcionamento adequado.

- A **Adaptação Adaptativa**(*Adaptive Adaptation*) ocorre em resposta a mudanças no contexto que podem afetar o comportamento da aplicação. Por exemplo, mudanças no sistema operacional ou no *hardware* no qual o sistema está em execução podem exigir essa adaptação.

- **Adaptação Perfectiva** (*Perfective Adaptation*) tem como objetivo aprimorar uma aplicação, mesmo que ela esteja funcionando corretamente. Por exemplo, uma aplicação pode encontrar um novo componente que desempenhe o mesmo papel de um que já está em uso, porém de forma mais rápida e consumindo menos memória, e trocar um pelo outro.

- Por fim, a **Adaptação de Extensão** (*Extending Adaptation*) ocorre em função de novas funcionalidades necessárias para a aplicação e que não tenham sido consideradas no tempo de desenvolvimento. Nesse caso, a aplicação deve ser estendida através de novos componentes que provejam tais funcionalidades.

Outro critério comumente utilizado para categorizar a adaptação é o **momento em que ela ocorre**. De acordo com esse critério, McKinley et al. (2004) classifica a adaptação como estática ou dinâmica. A adaptação estática ocorre em tempo de desenvolvimento, compilação ou carregamento. Entre seus objetivos estão a maximização do reúso de código e a facilitação da manutenção do *software* (KAKOUSIS et al., 2010). A adaptação estática pode ser dividida em

três tipos. O primeiro se chama *hardwired*. Ele ocorre quando a adaptação é realizada em tempo de desenvolvimento do programa e não pode ser alterada, exceto com uma nova codificação. O segundo tipo, chamado *customizable*, requer apenas a recompilação ou a ligação com um novo ambiente. O último, *configurable*, posterga a decisão final de adaptação para o momento onde a aplicação em execução carrega o componente correspondente. Ele oferece maior dinamismo do que os dois métodos citados anteriormente, mas ainda assim é considerado estático.

A adaptação dinâmica é uma abordagem mais flexível e ocorre em tempo de execução. Uma parte do programa pode ser substituída, retirada, adicionada, estendida ou alterada em um programa já em execução sem desligá-lo ou reiniciá-lo. Este tipo de adaptação é comumente requerido em aplicações móveis e sensíveis ao contexto, devido a necessidade de adaptação dinâmica a mudanças no contexto (KAKOUSIS et al., 2010). A adaptação dinâmica pode ser dividida em: *tunable software* e *mutable software*. A primeira não permite que a lógica de negócio seja alterada. A segunda permite que a lógica seja alterada possibilitando que a função do aplicativo até mesmo mude através da adaptação. Apesar de muito poderosa a complexidade para manutenção da integridade do sistema é alta nesse último tipo de adaptação. A Figura 2.5 apresenta uma comparação entre as adaptações estáticas e dinâmicas. Nela, a adaptação *hardwired* é considerada a de menor dinamismo, enquanto no outro extremo são encontradas as do tipo *tunable* e *mutable*.

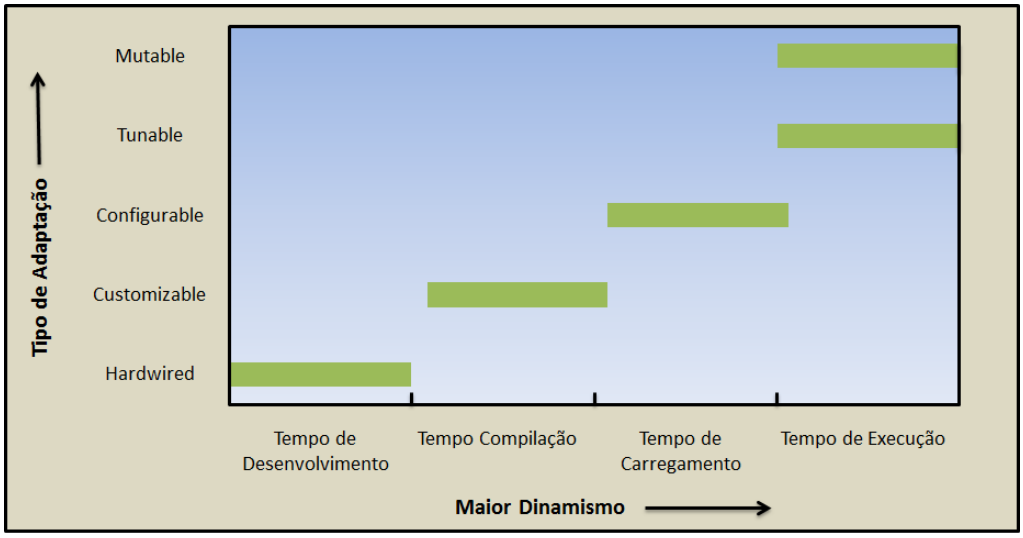

Figura 2.5: Comparação entre adaptações estáticas e dinâmicas. Adaptado de (MCKINLEY et al., 2004).

(KAKOUSIS et al., 2010) afirmam que a partir de uma **perspectiva técnica** a adaptação pode ser definida como baseada em parâmetros ou composicional. A adaptação baseada

em parâmetros consiste na adaptação por meio de mudanças de valores em propriedades ou variáveis sem mudança estrutural no sistema. Já a adaptação composicional permite modificação na estrutura do *software* por meio da adição e remoção de componentes. A adaptação composicional é uma ferramenta importante em sistemas móveis e sensíveis ao contexto, pois permite melhor gerenciamento dos seus recursos limitados (KAKOUSIS et al., 2010). Quando a adaptação composicional ocorre dinamicamente, ela pode ser realizada por meio da inicialização, parada ou troca de componentes já presentes e instalados no dispositivo ou pela instalação de componentes completamente novos (usualmente obtido de repositórios remotos) no sistema. Quando novos componentes são obtidos e instalados em tempo de execução, essa adaptação também pode ser classificada como implantação dinâmica (FONSECA, 2009). A implementação da adaptação composicional de um *software*, em geral é baseada em alguma forma de especificação formal da arquitetura do sistema. Entre as abordagens de especificação mais utilizadas estão as baseadas em grafos, álgebra de processos e em lógica (BRADBURY et al., 2004).

Ainda na perspectiva técnica, Salehie e Tahvildari (2011) destacam que a adaptação pode fazer uso da programação orientada a aspectos (AOP) e da arquitetura orientada a serviços (SOA). A AOP ajuda no encapsulamento de interesses (*concerns*) na forma de aspectos e na implementação de adaptações com níveis de granularidade mais alta que os atingidos por meio da utilização de componentes. Já a SOA suporta a adaptação de *software* facilitando a composição de serviços fracamente acoplados.

O *framework* de aquisição de contexto proposto nesse trabalho de dissertação possui dois tipos de adaptação. Uma delas é adaptativa, estática e composicional, devido ao fato de que é possível determinar quais componentes de aquisição de contexto devem ser implantados em cada dispositivo com suas peculiaridades próprias de *hardware* e *software*. A outra é de extensão, dinâmica e também composicional, pois componentes de aquisição de contexto podem ser inseridos ou removidos do *framework* conforme mudanças de requisitos em tempo de execução.

2.4 Engenharia de Software Baseada em Componentes

2.4.1 Definição

A adaptação composicional, como descrito anteriormente, utiliza do paradigma de componentes de *software* para seu funcionamento. Várias definições sobre o que é um componente já foram propostas na literatura. Uma das mais citadas é a de Szyperski et al. (2002) que afirma que um componente de *software* é uma unidade de composição com interfaces contratuais especificadas, dependências explicitas de contexto apenas. Além disso, componentes são implantáveis independentemente e estão sujeitos a composição por terceiros. Mas recentemente, Wang e Qian (2005) definem componente de *software* como um pedaço auto contido e auto implantável de código com uma funcionalidade bem definida e que pode ser ligado com outros componentes através de uma interface.

Alguns trabalhos utilizam o conceito de modelo de componentes para definir o que é um componente. Por exemplo, Heineman e Councill (2001) afirmam que um componente de *software* é um elemento de *software* que está em conformidade com um modelo de componentes e pode ser independentemente implantado e composto sem modificação de acordo com um padrão de composição. Heineman e Councill (2001) definem ainda que um modelo de componente é um conjunto de padrões para implementação, nomeação, interoperabilidade, personalização, composição, evolução e implantação de um componente. Lau e Wang (2007) afirmam que um modelo componentes é uma definição da semântica, sintaxe e formas de composição de componentes. A semântica diz o que um componente é e a sintaxe como esses componentes são definidos, construídos e representados. A forma de composição especifica como tais componentes são conectados e compostos entre si para a criação de novos componentes ou de sistemas. Nesse trabalho foi utilizada a definição componente de Lau e Wang (2007) na criação dos componentes de aquisição de contexto.

A utilização de componentes para construção de sistemas pode ser chamada Engenharia de Software Baseada em Componentes (*Component Based Software Engineering* ou CBSE). A CBSE busca acelerar o desenvolvimento de *software* e reduzir custos a partir do reúso de componentes previamente desenvolvidos, atividade classificada como desenvolvimento com reúso. Por outro lado, para que esses componentes estejam disponíveis, é necessário que alguém os desenvolva, essa atividade é chamada de desenvolvimento para reúso. O nível de reúso

alcançado pela utilização de componentes é considerado altos (WANG; QIAN, 2005), pois componentes permitem vários tipos de reúso como caixa-preta, caixa-cinza e caixa-branca. O tipo caixa-preta ocorre quando o código do componente é disponibilizado e pode ser estudado, adaptado ou modificado. Caixa-preta, por sua vez, não permite qualquer acesso ao código interno do componente. O tipo caixa-cinza é quando o código de um componente está parcialmente disponível. Outra característica importante da CBSE é que ela permite um bom gerenciamento de mudanças em um *software*, pois componentes são fáceis de adaptar a mudanças ou a novos requisitos.

Dentro do cenário da CBSE, diversos *frameworks* de componentes foram desenvolvidos para auxiliar o desenvolvimento de sistemas orientados a componentes. Alguns exemplos deles são: CORBA (GROUP, 1995), COM (WILLIAMS; KINDEL, 1994), EJB[1] e OSGi (ALLIANCE, 2003). Nessa dissertação, foi adotado o *framework* de componentes OSGi, detalhado a seguir. Essa escolha deveu-se ao fato dele possibilitar adição, remoção ou modificação de componentes até mesmo em tempo de execução e de ser baseado em Java, uma linguagem multiplatafoma.

2.4.2 OSGi

Open Service Gateway initiative (OSGi) é um conjunto de especificações abertas, criadas por um consórcio de industrias parceiras, que definem componentes de *software* dinâmicos para Java. Essas especificações possuem diversas implementações como as dos projetos Apache Felix[2] e Equinox[3]. Ao contrário da maioria dos *frameworks* de componentes existentes, aqueles baseados nas especificações do OSGi possuem uma arquitetura flexível que permite que um sistema evolua a partir da adição, remoção ou modificação de componentes em tempo de execução ou não (CRNKOVIC et al., 2011).

No OSGi, os componentes de *software* básicos para a criação de sistemas são chamados *bundles*. Tais componentes são compostos por um arquivo JAR (*Java Archive*) que contem código binário e um arquivo manifesto. Este arquivo declara, entre outras informações, pacotes Java exportados e importados pelo componente, que funcionam respectivamente como interfaces oferecidas e requeridas.

[1] http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[2] http://felix.apache.org/
[3] http://www.eclipse.org/equinox/

Conforme citado anteriormente, o OSGi permite a adição e remoção de componentes até mesmo em tempo de execução, porém todo o ciclo de vida de um *bundle* é controlado diretamente pelo *framework*. Tal característica facilita o desenvolvimento de sistemas com adaptação composicional. O ciclo de vida de um *bundle* consiste de seis estados, conforme visto na Figura 2.6:

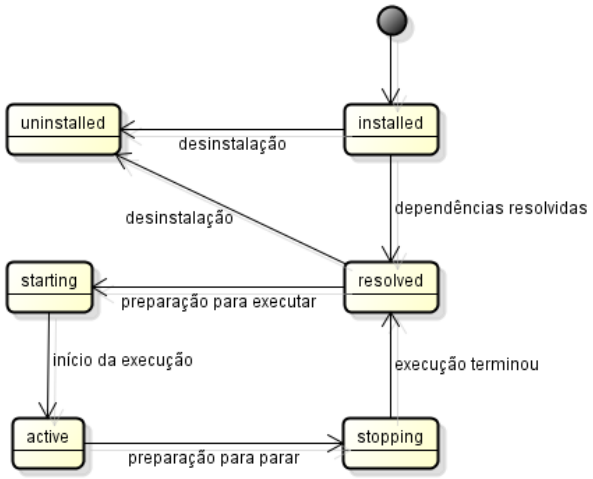

Figura 2.6: Ciclo de vida de um bundle.

- **uninstalled**: ocorre quando um *bundle* está desinstalado e não pode ser usado;
- **installed**: o *bundle* está instalado, mas não está pronto para iniciar o funcionamento, pois suas dependências não estão resolvidas;
- **resolved**: nesse estado o *bundle* está instalado e pronto para iniciar o funcionamento;
- **starting**: esse estado ocorre quando *bundle* está se preparando para iniciar a execução;
- **active**: o *bundle* está sendo executado; e
- **stopping**: o *bundle* está em processo de parada.

Para a criação de um *bundle* é necessário criar uma classe que implemente a interface Java org.osgi.framework.BundleActivator. A interface BundleActivator possui em suas assinatura os métodos *start* e *stop* que são chamados quando o componente inicia o funcionamento e quando ele para. É necessário também a criação de um arquivo manifesto declarando

alguns metadados sobre o componente como o nome do componente e os pacotes que ele importa e exporta. Por fim, um arquivo JAR contendo os executáveis do código Java e o manifesto deve ser gerado.

Existem dois tipos de conexão entre componentes no OSGi (WANG; QIAN, 2005): estáticas através de importação/exportação de pacotes, coleção de classes, e dinâmicas através de serviços. A conexão por pacote gera uma dependência estática que deve ser definida no momento da criação do *bundle*. Essa conexão pode ser de importação, onde o *bundle* necessita daquele pacote de classes para funcionar, ou de exportação, quando ele oferece um pacote para outros componentes. A conexão dinâmica ocorre por meio de serviços fornecidos por *bundles*. Um serviço deve ser registrado no *framework* para se tornar acessível para outros *bundles*. É possível a um *bundle* oferecer vários serviços ou nenhum.

A adaptação composicional permitida pelo OSGi bem como o suporte ao gerenciamento do ciclo de vida dos bundles, permite que uma infraestrutura de gerenciamento de contexto construída sob ele usufrua de muitos benefícios. Em primeiro lugar, caso a arquitetura da infraestrutura seja bem projetada, o problema do tudo-ou-nada pode ser eliminado. Além disso, a adaptação dinâmica suportada por ele, juntamente com seu gerenciamento de ciclo de vida, permite uma economia de recursos por meio do desligamento de componentes que não estão em uso. Por fim, a possibilidade de adicionar componentes em tempo de execução também facilita o gerenciamento de requisitos das aplicações, pois novos componentes podem ser adicionados sob demanda. Apesar dos benefícios, algumas limitações também são imposta pelo uso do OSGi. Por exemplo, só existe suporte para linguagem Java.

2.5 Conclusão

Neste capítulo foram apresentados os conceitos de sensibilidade ao contexto, adaptação na computação móvel e Engenharia de Software Baseada em Componentes (CBSE). Sistemas sensíveis ao contexto tem a capacidade de oferecer serviços e funcionalidades de acordo com o contexto onde estão inseridos. Várias infraestruturas de suporte a criação desses tipo de sistema já foram propostas. Em geral, elas buscam auxiliar na aquisição, enriquecimento e gerenciamento de informações contextuais e muitas vezes proveem mecanismos de suporte a adaptação ao contexto por parte das aplicações.

Dentre as técnicas de adaptação para a computação móvel e sensível ao contexto apresentadas, destacou-se a composicional, que permitem modificação estrutural no software por meio da adição e remoção de componentes em tempo de implantação (estática) ou execução (dinâmica). Essa adaptação, quando aplicada a própria infraestrutura de suporte, permite enfrentar o problema do tudo ou nada descrito na Seção 1.2 e alcançar uma economia de recursos do dispositivo. Além disso, com a adaptação composicional dinâmica é possível remover ou parar componentes que não estejam sendo utilizados em um dado momento, acarretando em mais economia de recursos.

A adaptação composicional faz uso do paradigma da CBSE, que divide o software em unidades de composição (componentes) reutilizáveis e com interfaces e funcionalidades bem definidas. *Frameworks* de componentes buscam facilitar o desenvolvimento baseado na CBSE. Em particular, os que implementam as especificações OSGi, permitem a instalação, remoção, inicialização ou parada de componentes até mesmo em tempo de execução.

O próximo capítulo apresenta as principais infraestruturas relacionadas com o *framework* proposto e realiza uma comparação entre eles.

3 PRINCIPAIS INFRAESTRUTURAS E TRABALHOS RELACIONADOS

Neste capítulo são apresentadas algumas das principais infraestruturas de suporte a criação de sistemas sensíveis ao contexto e os principais trabalhos relacionados ao *framework* CAM. Na Seção 3.1 são descritas algumas das principais infraestruturas de suporte a criação de sistemas sensíveis ao contexto e também o sistema SysSU, que foi adaptado e utilizado neste trabalho de dissertação. A Seção 3.2 expõe trabalhos relacionados ao *framework* CAM, que são infraestruturas projetadas para uso embarcado em dispositivos móveis ou que apresentam alguma adaptação de *software* para economia de recursos.

3.1 Infraestruturas de Suporte

Várias infraestruturas de suporte para a construção de sistemas sensíveis ao contexto foram criadas ou propostas para facilitar o desenvolvimento e aprimorar o reúso e a extensibilidade desse tipo de aplicações, como pode ser visto nos *surveys* (BALDAUF et al., 2007), (KELING et al., 2012) e (HONG et al., 2009). Nas próximas subseções são apresentadas algumas infraestruturas encontradas na literatura juntamente com suas arquiteturas.

3.1.1 Context Toolkit

Context Toolkit (DEY et al., 2001) é um *framework* que separa a aquisição e a representação de contexto da entrega às aplicações e da forma como as mesmas reagem ao contexto. Esta é uma infraestrutura distribuída que utiliza comunicações ponto a ponto entre sensores e computadores. Quatro tipos principais de componentes são descritos em sua arquitetura: *Context Widget*, *Interpreter*, *Aggregator*, *Service* e *Discoverer*. A Figura 3.1 mostra um exemplo de configuração válida dos componentes da infraestrutura.

Context widgets são responsáveis por encapsular sensores provendo uma interface uniforme de acesso a contexto e escondendo a complexidade de interação com sensores reais. A comunicação com essas entidades pode ser realizada síncrona ou assincronamente. Eles são responsáveis também por guardar um histórico do contexto capturado. Apesar do comportamento padrão dos *context widgets* não possibilitar o acesso a detalhes sobre o uso dos sensores, é possível ter acesso a informações como tipo de sensor utilizado e precisão da coleta de dados.

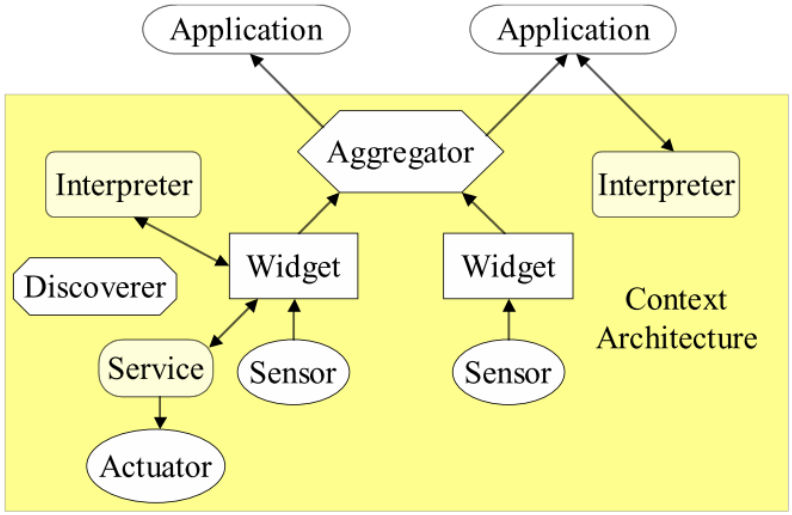

Figura 3.1: Exemplo de configuração válida dos componentes do Context Toolkit (DEY et al., 2001).

Context Widgets são executados de forma independente de uma aplicação individual e podem ser usados por várias simultaneamente.

Entidades do tipo *Interpreter* elevam o nível de abstração de informações contextuais. Como mencionado na Seção 2.2.1, aplicações sensíveis ao contexto usualmente requerem informações mais abstratas do que os sensores disponibilizam. Para gerar essas informações, uma ou mais fontes de contexto podem ser utilizadas. Todos os *interpreters* também possuem uma interface comum.

Aggregators são responsáveis por coletar contextos que estão logicamente relacionados em um repositório comum. Sua maior importância se deve ao fato de que contextos podem ser oriundos de sensores distribuídos. Várias aplicações podem acessar informações de um único *aggregator*. Mudanças no contexto de *aggregators* podem ser acessadas de forma síncrona ou assíncrona. Um histórico de contexto referente àquele *aggregator* também está disponível para aplicações.

Services são entidades que realizam tarefas para uma ou mais aplicações. Eles são responsáveis por controlar ou alterar informações de estado do ambiente utilizando um atuador. As formas de interação com *services* podem ser síncronas ou assíncronas.

Entidades do tipo *discoverer* são responsáveis por manter um registro de quais funcionalidades estão disponíveis no *framework*. Todas as outras entidades citadas anteriormente notificam sua presença nesse registro quando são iniciadas. Informações de como interagir com

elas também estão incluídas. Aplicações podem utilizar *discovers* para encontrar entidades com um nome ou tipo específico. Outra opção é realizar uma busca através de seus atributos ou serviços. Assim como os *widgets*, *discoverers* são capazes de notificar aplicações sobre o aparecimento ou desaparecimento de um componente no *framework*.

3.1.2 CASS

CASS (FAHY; CLARKE, 2004) é um *middleware* com arquitetura cliente-servidor que oferece suporte a aplicações móveis sensíveis ao contexto. Entre suas principais características estão o suporte a informações contextuais de alto nível semântico e a separação de inferências contextuais do código da aplicação. A arquitetura de CASS consiste de um servidor, dispositivos móveis, que são clientes, e computadores dotados de sensores, chamados de nós sensores, que alimentam o servidor com dados contextuais, como mostra a Figura 3.2. As principais entidades encontradas no servidor são um banco de dados, que armazenará, entre outras coisas, o contexto da aplicação e um motor de inferência para descoberta de informações contextuais de mais alto nível. A localização dessas entidades permite que a maioria das restrições de armazenamento, memória e processamento comuns a dispositivos móveis sejam ignoradas, pois ocorrem no servidor. Os computadores dotados de sensores funcionam como fonte de informações contextuais para o servidor. Os dispositivos móveis (clientes) são, por sua vez, consumidores dos serviços prestados pelo servidor.

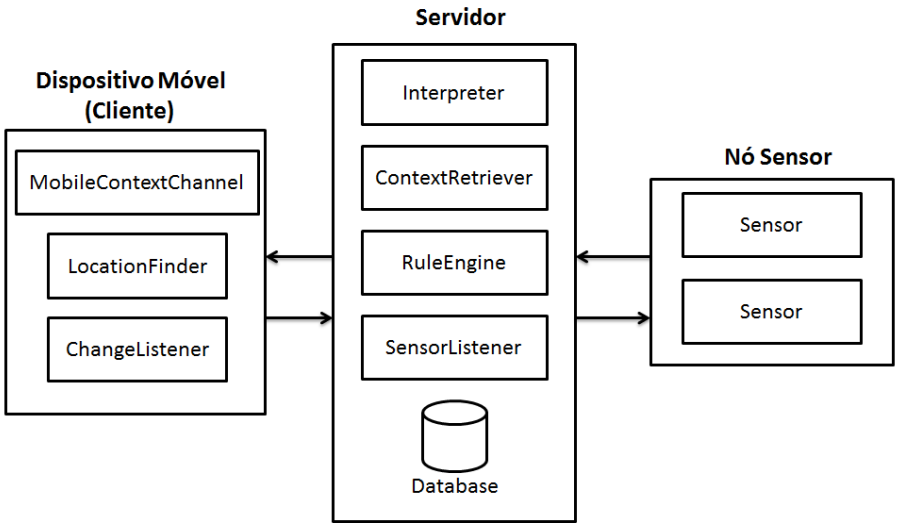

Figura 3.2: Arquitetura do middleware CASS. Adaptado de (FAHY; CLARKE, 2004).

Dentro dos nós sensores são encontrados apenas um tipo de componente chamado

de *sensor*. Componentes desse tipo são responsáveis por recuperar dados contextuais e enviá-los para o servidor por meio da rede. Dentro do servidor são encontrados cinco tipos de entidades: *Interpreter*, *ContextRetriever*, *RuleEngine*, *SensorListener* e *Database*. *SensorListeners* escutam por atualizações de contexto geradas pelos nós sensores e armazenam essas informações no *Database*. *ContextRetrievers* recuperam essas informações armazenadas. Tanto *SensorListeners* como *ContextRetrievers* podem utilizar serviços de um *Interpreter*, que eleva o nível semântico de informações contextuais. Por fim, *RuleEngine* oferece características como enriquecimento de informações contextuais e geração de eventos quando determinados contextos são atingidos. Dentro de computadores clientes, são encontrados os componentes *ChangeListener* e *LocationFinder*. O componente *ChangeListener* escuta atualizações contextuais que ocorrem nos servidores.

Apesar dos benefícios de maiores recursos computacionais do servidor, tal abordagem traz uma dependência completa do acesso à rede. Essa característica pode diminuir consideravelmente o tempo de bateria dos dispositivos e dificultar o acesso aos serviços devido à instabilidade das redes existentes na computação móvel. Para atenuar esse problema, o *middleware* CASS utiliza o conceito de *caching*. A arquitetura de CASS apresenta uma clara distinção entre os dispositivos móveis (clientes) e aqueles dotados de sensores. Essa separação tem se tornado distante da realidade, visto que os dispositivos móveis recentes apresentam uma série de sensores embutidos.

3.1.3 SOCAM

O trabalho de Gu et al. (2005) propõe um *middleware* distribuído sensível ao contexto chamado SOCAM. Ele é orientado a serviço e provê suporte para aquisição, descoberta, interpretação e acesso a vários contextos para a construção de serviços sensíveis ao contexto. O *middleware* proposto consiste de serviços que agem de forma independente e podem ser dos seguintes tipos: *Context Provider*, *Context Interpreter*, *Context Database*, *Context-Aware Services* e *Service Locating Service*. Uma visão geral da arquitetura pode ser vista na Figura 3.3

Context providers encapsulam sensores e proveem informações de mais alto nível semântico do que sensores físicos comuns seriam capazes. Eles são separados em dois tipos: externos e internos. O primeiro obtém contexto de fontes externas como informações de clima.

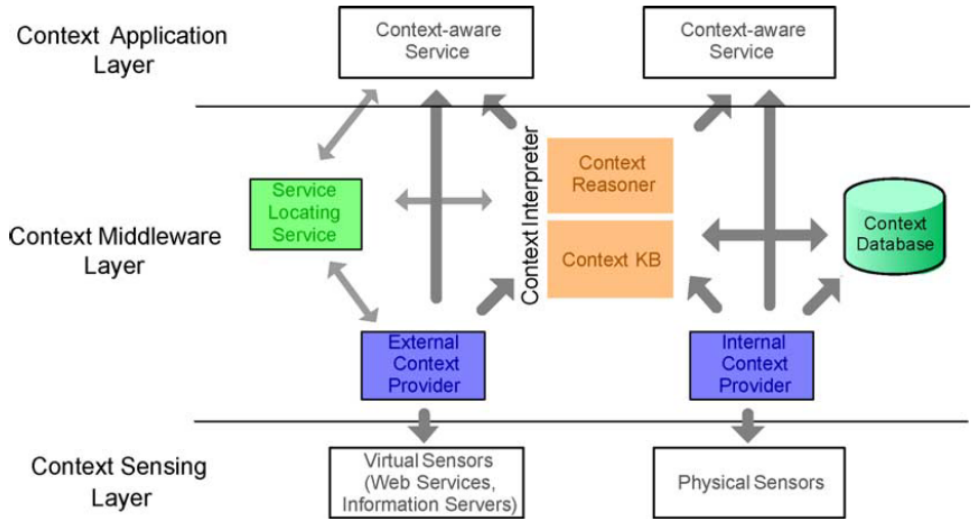

Figura 3.3: Arquitetura do SOCAM (GU et al., 2005).

O segundo captura o contexto diretamente de sensores espalhados em um ambiente, como casa ou carro inteligente. Informações contextuais geradas por *context providers* são disponibilizadas como eventos no formato de ontologias.

Context interpreter é um componente que realiza inferências a partir de informações contextuais de mais baixo nível. Ele também realiza consultas de informações contextuais, mantem sua consistência e resolve seus conflitos. Internamente, ele consiste em um motor de inferência de contextos e uma base de conhecimento. Vários motores podem ser incorporados ao *context interpreter* e novas regras de inferência também podem ser especificadas. A base de conhecimento possui uma interface que possibilita a outros serviços realizar consultas, adição ou modificação de informações contextuais. Para garantir que essas informações sejam recentes, atualizações podem ser realizadas nos dados.

Context-aware services possuem um conjunto de regras que dispara ações adequadas para certos tipos de contexto. As informações contextuais são obtidas através de consultas diretas (modo síncrono) realizadas em *context providers* ou da escuta de eventos (modo assíncrono) também gerados por *context providers*.

S*ervice locating service* é um componente que permite aos *context providers* e *context interpreters* registrarem sua presença. Aplicações podem, então, encontrar os serviços disponíveis pelo *framework*.

3.1.4 Multilayer Framework

Bettini et al. (2010) propõem uma infraestrutura denominada Multilayer Framework mais focada em como informações contextuais são modeladas e em prover informações contextuais de alto nível semântico para aplicações. Esse *framework* é dividido em três camadas, como é mostrado na Figura 3.4.

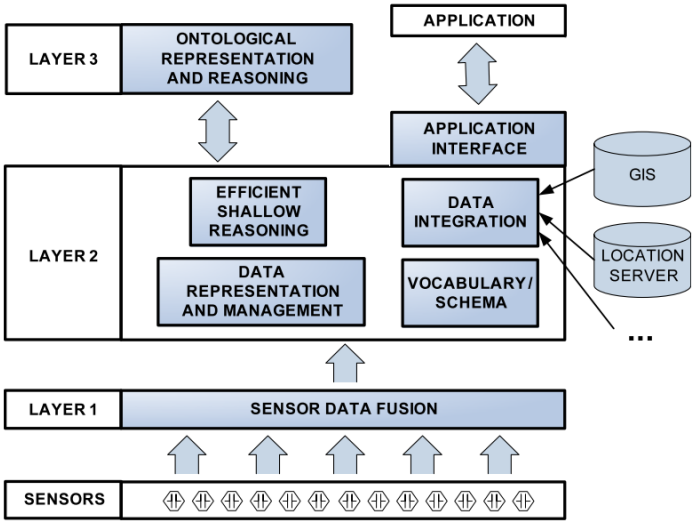

Figura 3.4: Multilayer Framework (BETTINI et al., 2010).

A primeira camada adquire, processa e propaga informações contextuais de baixo nível em um ambiente rico em sensores. A segunda camada possui uma representação simples de informações contextuais que permite que inferências eficientes sejam realizadas. Ela também possui como funcionalidade recuperar informações contextuais de fontes externas. A terceira camada possui uma ontologia que representa o contexto e também é utilizada para a realização de inferências. Entre os principais objetivos dessa camada está a especificação da semântica dos termos de contexto, que é importante para compartilhamento da informação, e checagem de consistência do modelo de contexto.

3.1.5 SysSU

SysSU (LIMA et al., 2011) é uma infraestrutura que provê suporte a implementação de sistemas ubíquos. Suas funcionalidades permitem que informações contextuais em forma de tuplas sejam armazenadas e acessadas de forma síncrona e assíncrona, característica atingida pela composição das abordagem de espaço de tuplas (CARRIERO; GELERNTER, 1989) e de

publicação/subscrição (EUGSTER et al., 2003). SysSU atua apenas como a camada de Armazenamento/Gerenciamento descrita por Baldauf et al. (2007), não possuindo funcionalidades como aquisição de contexto ou enriquecimento de informações contextuais.

Todo o funcionamento dessa infraestrutura é baseado em tuplas. Uma tupla é uma composição de um conjunto de campos chave/valor, por exemplo, {(user,“John”), (age,10), (gender,“M”)}. Existem duas formas para que aplicações possam utilizar informações publicadas no SysSU. A primeira é realizar uma consulta direta o que exige que as informações tenham sido disponibilizadas nele antes da consulta ser realizada. A segunda forma é a baseada em eventos (publicação/subscrição), que é utilizada quando a informação ainda não está disponível, mas uma aplicação deseja ser notificada quando estiver. As duas formas de acesso a informações fazem uso de *templates* e *filtros* para selecionar as tuplas desejadas.

Um *template* é uma coleção de campos representando um conjunto ou subconjunto de campos que compõem uma tupla requerida. Por exemplo, para encontrar uma tupla com um campo contendo a chave “user” o seguinte *template* poderia ser utilizado {(user,?)}. Um *template* também pode ser utilizado para retornar tuplas com valores específicos para uma chave. Por exemplo, o *template* {(user,?),(age,10)} retorna todas as tuplas que contenham qualquer valor para o campo com chave “user” e que também contenham um campo com chave “age” e valor exatamente igual a 10 anos de idade. O uso de *templates* permite realizar apenas comparações de igualdade de valores. Para melhorar a expressividade da seleção de tuplas, um *filtro* deve ser utilizado. *Filtros* são criados utilizando JavaScript e são limitados apenas pela própria expressividade da linguagem. A Figura 3.5 mostra um exemplo de filtro que seleciona tuplas onde um campo com chave “age” possui valores maiores do que 18.

```
function filter(tuple) {
    if (tuple.age >= 18)
        return true;
    else
        return false;
}
```

Figura 3.5: Exemplo de implementação de filtro.

Toda vez que uma seleção ocorre, o *template* é primeiramente utilizado para selecionar um subconjunto de todas as tuplas contidas no SysSU. A partir desse ponto, caso exista um filtro, cada tupla desse subconjunto será submetida ao filtro. Toda tupla que passar pelo método

filter e gerar um retorno *true* passa com sucesso pelo filtro.

A arquitetura dessa infraestrutura é originalmente cliente/servidor. No servidor encontra-se o UbiCentre, um repositório de espaços de tuplas acessado por clientes magros que podem adicionar ou ler tuplas utilizando o UbiBroker. A Figura 3.6 mostra a arquitetura da infraestrutura.

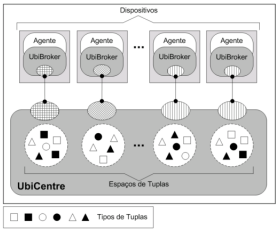

Figura 3.6: Arquitetura do SysSU. Adaptado de (LIMA et al., 2011).

O SysSU é uma ferramenta eficiente para promover o desacoplamento entre aplicações e serviços que utilizam o espaço de tuplas e para coordenar suas operações. Durante esse trabalho de dissertação sua arquitetura foi adaptada para execução totalmente embarcada em um único dispositivo. Após a adaptação, ele foi utilizado em conjunto com o *framework* de aquisição de contexto proposto para prover meios sofisticados de acesso as informações adquiridas.

3.2 Trabalhos Relacionados

Apesar de várias infraestruturas de suporte a sistemas sensíveis ao contexto já terem sido propostas e implementadas, como visto na Seção 3.1, muitas dessas delas não são projetadas para uso em dispositivos móveis devido as limitações de recursos como processamento e principalmente de bateria limitada desses dispositivos. Além disso, essas infraestruturas em geral consideram que os sensores que proveem informações contextuais estão espalhados pelo ambiente. Por fim, também é comum que elas façam uso de servidores para processamento e armazenamento de informações contextuais.

Nesse cenário, foram considerados como trabalhos relacionados ao *framework* CAM infraestruturas que atendessem pelo menos um de dois critérios:

1. a infraestrutura é direcionada para utilização embarcada em dispositivos móveis ricos em sensores embarcados, tais como *smartphones* e *tablets*; ou

2. existe algum tipo de adaptação da infraestrutura buscando otimizar a utilização de recursos do dispositivo.

Nas próximas subseções são apresentadas infraestruturas baseadas nesses critérios.

3.2.1 MobiCon

Mobicon (LEE et al., 2012) é um *framework* de monitoramento de contexto que deve ser capaz de lidar com problemas e desafios de recursos em ambientes móveis ricos em sensores. Sua arquitetura consiste em uma parte central embarcada em um DM que se comunica com uma série de sensores externos ao dispositivo e espalhados em um ambiente ubíquo. Essa infraestrutura possui uma abordagem para gerenciamento dos sensores que busca minimizar a quantidade em funcionamento baseado no interesse das aplicações (expressos por meios das consultas).

O MobiCon permite que várias aplicações o utilizem concorrentemente requisitando informações contextuais de interesse. Tais requisições são realizadas através de registro de consultas de alto nível chamadas *Context Monitoring Query* (CMQ). As CMQ serão posteriormente traduzidas para uma expressão lógica de baixo nível que envolva diretamente sensores, por exemplo a CMQ "temperature == hot" poderia ser traduzida para "thermometer > 86". A partir da expressão de baixo nível e de regras de expressões booleanas o *framework* encontra um conjunto mínimo de sensores necessários para avaliar as CMQ. Apenas esse conjunto mínimo de sensores necessários é utilizado, enquanto os outros sensores são desativados, possibilitando economia de recursos do DM.

O fato do funcionamento do MobiCon ser baseado em consultas de alto nível, não permite que aplicações que o utilizem tenham acesso e possam monitorar diretamente os valores de contexto de baixo nível. Por exemplo, uma aplicação não é capaz de realizar uma leitura direta e continua do termômetro em graus, mas apenas de saber se esta temperatura é considerada alta (*hot*) ou não. Além disso, MobiCon não possibilita implantação dinâmica de componentes.

3.2.2 ContextProvider

ContextProvider (MITCHELL et al., 2011) é um *framework* que integra informações contextuais coletadas por um *smarphone* e provê uma interface de consulta para aplicações

de monitoramento médico do dispositivo possam acessá-las. Sua arquitetura consiste em uma parte central presente no DM que recupera informações contextuais originadas de sensores físicos embutidos no próprio DM ou de *webservices*.

Esse *framework* possui uma abordagem de ajuste de frequência de leituras efetuadas pelos sensores para alcançar um melhor gerenciamento de energia e menor consumo do espaço de armazenamento de dados do DM. Quando o ContextProvider detecta que pouca ou nenhuma alteração está sendo sensoreada por um determinado sensor, então a frequência de leitura do mesmo é diminuída. Por outro lado, mudanças no contexto também podem reverter a frequência de leitura para uma mais elevada. Além do gerenciamento da frequência de leituras, alguns sensores também podem ser substituídos por outros que consomem menor quantidade de recursos do DM (e.g., GPS por posicionamento baseado em triangulação de antenas).

Outra característica importante dessa infraestrutura é sua extensibilidade. Para facilitar a adição de novos sensores, ContextProvider utiliza APIs e padrões existentes e conhecidos em sua construção. O *framework* não permite, todavia, extensão de sua estrutura em tempo de execução por meio de implantação dinâmica nem adaptação composicional dinâmica para a maioria dos seus componentes.

3.2.3 Kaluana

Kaluana (FONSECA, 2009) é um *middleware* para DM dotados de Android que define um modelo de componentes orientado a serviços. Esse modelo permite composição, reconfiguração e implantação de componentes em tempo de execução. Essa infraestrutura é baseada nos princípios de CBSE, apresentada na Seção 2.4, e possibilita adaptação composicional de aplicações em tempo de desenvolvimento ou em tempo de execução em função do contexto.

O funcionamento de Kaluana é baseado em três entidades principais: gerenciador de adaptações, gerenciador de componentes e repositório de componentes. A primeira é responsável por analisar possíveis adaptações baseadas em mudanças no contexto enquanto a segunda por ativar e desativar componentes de acordo com as requisições geradas pelo gerente de adaptações. O repositório de componentes mantém uma referência para todos os componentes disponíveis no DM. Caso uma requisição de componentes não seja satisfeita pelo repositório, o gerenciador de componentes realiza uma busca em um repositório remoto.

Apesar de facilitar a adaptação ao contexto, a aquisição do contexto, em si, não possui nenhum suporte do *middleware* e é realizada através de acesso direto aos sensores do dispositivo. Além disso, o modelo de componentes do Kaluana estende o modelo de componentes do Android, por isso para que novos componentes sejam implantados dinamicamente, é necessário que o usuário confirme a operação através da instalação de um APK (arquivo que contem componentes e/ou aplicativos Android). Sendo assim, a instalação de novos componentes não é uma tarefa transparente para o usuário.

3.2.4 Middleware de Preuveneers e Berbers

Preuveneers e Berbers (2007) também propõem um *middleware* para DM capaz de realizar adaptação composicional estática e dinâmica para melhor utilização de seus recursos. Esse *middleware* é composto de 3 camadas adaptáveis: *application layer*, onde se encontram as aplicações; *context layer*, responsável por adquirir informações contextuais; e *runtime layer*, que provê funcionalidades básicas para a execução de componentes.

Os tipos de componentes presentes na camada de contexto, de maior valor para este estudo, são: *input*, *filter*, *persistency*, *transformation* e *reasoning*. O componente do tipo *input* encapsula sensores físicos e virtuais. O tipo *filter* filtra dados irrelevantes ou velhos e compara precisão de diferentes *inputs*. O componente *persistency* provê um repositório para informações contextuais e um modelo de contexto. *Transformation* é responsável por transformar uma representação de dados em outra (e.g., de posição em latitude e longitude para cidade onde usuário se encontra). Por fim, *reasoning* combina dados contextuais para geração de novas informações.

A adaptação da camada de aquisição de contexto (*context layer*) dessa infraestrutura é dividida pelos seus criadores em dois tipos: de comportamento e estrutural. A de comportamento realiza adaptações como liberações de memória através da remoção de informações e de redução na frequência em que informações são lidas por sensores. A adaptação chamada de estrutural, é na verdade adaptação por composição. Ela funciona sobre dois tipos de componentes: *input* e *transformation*. Quando uma aplicação requer uma informação de contexto específica, os componentes apropriados para tal tarefa são adicionadas à cadeia de processos. Por outro lado, quando esta tarefa não é mais necessária, tais componentes são removidos. Apesar de permitir adaptação composicional, a implantação dinâmica de componentes não é suportada por esse *middleware*.

3.2.5 Comparação entre os Trabalhos

Todos os trabalhos relacionados apresentam relevância e foram utilizados como inspiração para a criação e desenvolvimento do *framework* proposto. O *framework* MobiCon, por exemplo, apresenta um *middleware* que busca encontrar um conjunto mínimo de sensores necessários para prover informações requisitadas por aplicações, consumindo o mínimo de energia possível. Através de adaptação composicional dinâmica, essa infraestrutura, então, desliga ou liga os sensores necessários para alcançar o conjunto mínimo. Entretanto, sua abordagem foca principalmente em sensores distribuídos em um ambiente, e não embarcados no dispositivo, e a interação entre as aplicações e o *framework* não permite leituras diretas aos valores fornecidos pelos sensores, como temperatura em graus.

O *framework* ContextProvider, por sua vez, não utiliza a estratégia do conjunto mínimo de sensores, mas busca alcançar uma eficiência energética através da alteração da frequência de leitura de seus sensores. Além disso, ele também possui um adaptação composicional dinâmica parcial, pois alguns poucos sensores podem ser substituídos em tempo de execução. Por exemplo, o sensor de localização por GPS pode ser substituído pelo de localização por triangulação de antenas.

Kaluana, é o trabalho que possui maior grau de adaptabilidade entre todos. Ele permite adaptação composicional estática e dinâmica e possibilita, ainda, implantação dinâmica de novos componentes. Todavia, o modelo de Kaluana é baseado no modelo da plataforma Android, que não permite uma implantação dinâmica transparente.

Preuveneers e Berbers (2007) também apresentam um *middleware* direcionado para DM e capaz de realizar adaptação composicional estática e dinâmica em toda a sua arquitetura. A implantação dinâmica, todavia, não é suportada pelo *middleware*.

A Tabela 3.1 apresenta uma comparação entre as trabalhos relacionados. Através dela, é possível analisar quais foram projetados para uso embarcado em um DM, se eles permitem instalação personalizada de componentes, reconfiguração composicional, implantação dinâmica e qual a principal localização dos sensores em cada infraestrutura.

Tabela 3.1: Comparação entre os trabalhos relacionados

	MobiCon	*Context Provider*	*Kaluana*	*Middleware de Preuveneers e Berbers*
Projetada para DM	Sim	Sim	Sim	Sim
Instalação Personalizada	--	--	Sim	Sim
Reconfiguração Composicional	Sim	--	Estática e Dinâmica	Estática e Dinâmica
Implantação Dinâmica	--	--	Intrusiva	--
Principais Localizações de Sensores	Ambiente Ubíquo	DM e webservices	--	DM e webservices

3.3 Conclusão

Este capítulo apresentou importantes infraestruturas de suporte a criação de sistemas sensíveis ao contexto. Todavia, essas infraestruturas não foram projetadas para utilização embarcada em um único dispositivo móvel, pois entre suas características estão uma arquitetura distribuída, onde sensores podem estar espalhados por um ambiente, e a utilização de servidores para processamentos como inferências e elevação de nível semântico de informações contextuas.

Nesse cenário, foram consideradas como trabalhos relacionadas, aquelas infraestruturas que se assemelhavam ao *framework* CAM por serem voltadas para utilização embarcada em DM dotados de vários sensores ou por apresentarem algum tipo de adaptação em busca de melhor gerenciamento de recursos do dispositivo. Os principais trabalhos encontrados foram: MobiCon, ContextProvider, Kaluana e um *middleware* proposto por Preuveneers e Berbers. Todos eles serviram como inspiração para a concepção do *framework* CAM.

O próximo capítulo apresenta o *framework* proposto, seu modelo de contexto e componentes. Também é apresentado o *middleware* LoCCAM, que faz uso do *framework* proposto.

4 O FRAMEWORK CAM

Este capítulo apresenta o *framework* proposto CAM. Uma visão geral do *framework* e de sua arquitetura é apresentada na Seção 4.1. A Seção 4.2 explica como uma informação contextual é representada pelo *framework* e qual a importância dessa representação na comunicação entre as aplicações e o *framework*. Na Seção 4.3, é apresentado com detalhes o que é um componente de aquisição de contexto e como um desenvolvedor pode criar o seu próprio. A adaptação estática e dinâmica do *framework* é explicada na Seção 4.4. Na Seção 4.5, é apresentado o LoCCAM, *middleware* onde o *framework* CAM foi inserido. Por fim, a Seção 4.6 apresenta outras contribuições deste trabalho decorrente do trabalho realizado no LoCCAM. Nela, são discutidos os principais mecanismos envolvidos no acesso a informações contextuais no LoCCAM.

4.1 Visão Geral

Assim como descrito no Capitulo 1, várias infraestruturas de suporte a criação de sistemas sensíveis ao contexto já foram propostas em diversos trabalhos, como pode ser visto em (BALDAUF et al., 2007) e (HONG et al., 2009). Essas infraestruturas podem auxiliar desde a aquisição de informação contextual até o processo de adaptação ao contexto que aplicações efetuam. Tais infraestruturas, todavia, em geral não são apropriadas para o uso embarcado em dispositivos móveis. Entre os principais motivos está a falta de adaptabilidade da infraestrutura em si (KELING et al., 2012), que pode ocasionar no consumo desnecessário de recursos do DM.

O objetivo principal deste trabalho é criar um *framework* para aquisição de contexto adaptável em dispositivos móveis dotados de sensores denominado CAM (Context Acquisition Manager). Esse *framework* permite uma implantação personalizada, através de adaptação composicional estática, para cada tipo de dispositivo, evitando assim o problema do tudo-ou-nada descrito na Seção 1.2. Da mesma forma, ele também possibilita a adaptação composicional dinâmica, que auxilia em um bom gerenciamento de recurso do DM e dos requisitos das aplicações que o estão utilizando. Para que o *framework* suportasse a adaptação composicional de sua estrutura, ele foi projetada utilizando o paradigma da Engenharia de Software Baseada em Componentes, descrita na Seção 2.4.

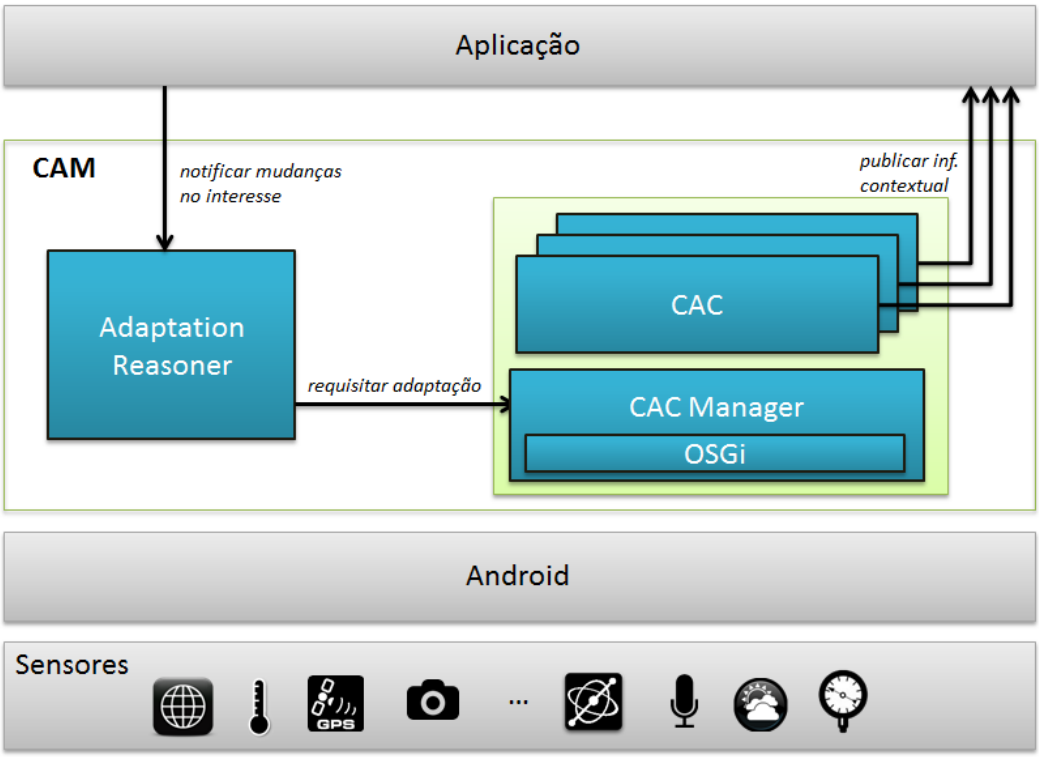

Figura 4.1: O *framework* CAM.

A Figura 4.1 mostra a arquitetura do *framework*. Para que aplicações o utilizem, elas devem mantê-lo atualizado com quais informações contextuais elas tem interesse de ser notificadas e quais informações elas não possuem mais interesse. O `Adaptation Reasoner` mantem uma lista contendo o interesse de todas as aplicações, que na prática representa a zona de interesse do sistema, assim como descrito na Seção 2.1. Sempre que a zona de interesse é modificada, o `Adaptation Reasoner` analisa a possibilidade de alguma adaptação para atender os novos interesses (requisitos) ou economizar recursos. Eventuais instruções de adaptação são então passadas ao `CAC Manager`. A unidade básica de composição e alvo da adaptação são os componentes de aquisição de contexto (`CAC`). Tais componentes são implementados em um *bundle* OSGi, o que facilita o gerenciamento do seu ciclo de vida.

O `CAC Manager` é o responsável por efetivamente iniciar ou parar o funcionamento de `CACs`. Ele encapsula todas as operações de gerenciamento de ciclo de vida de componentes que o OSGi oferece e disponibiliza uma interface simplificada para essa tarefa. Através dele, é possível instalar, desinstalar, iniciar e parar o funcionamento de um `CAC`. Também é ele quem disponibiliza a lista de todos os `CACs` disponíveis para adaptação ao `Adaptation Reasoner`. A Figura 4.2 apresenta a interface Java implementada pelo `CAC Manager`. Essa interface, além de garantir que certos métodos sejam implementados e disponibilizados publicamente, possibilita

que novas implementações do `CAC Manager` sejam criadas. Assim, o trabalho proposto pode ser estendido para utilizar outros *frameworks* de componente, ao invés do OSGi. Da mesma forma, o `Adaptation Reasoner` também possui uma interface que possibilita a implementação de um algoritmo de adaptação diferente, caso necessário.

```
public interface ICACManager {
        public void startCAC(Component cac);
        public void stopCAC(Component cac);
        public void installCAC(Component cac);
        public void uninstallCAC(Component cac);
        public Map<String, List<Component>> getListOfAvailableCACs();
}
```

Figura 4.2: Interface implementada pelo CAC Manager.

Considerando o processo de adaptação na computação ubíqua descrito por Kakousis et al. (2010), o `Adaptation Reasoner` é responsável pela fase de Planejamento da Adaptação, enquanto a Atuação da Adaptação é realizada pelo `CAC Manager`. Nesse caso, o que está sendo considerado como contexto para disparar adaptações é o interesse das aplicações.

4.2 Representaçao de Contexto

Para possibilitar a comunicação entre o *framework* e as aplicações, é necessária a utilização de um vocabulário compartilhado que represente tipos de informações contextuais. Cada tipo de informação contextual (e.g., temperatura, localização, clima) requer um nome único que possa identificá-la. Esse nome único deve ser utilizado por um `CAC` para determinar que tipo de informação ele provê e publica, e por aplicações, para consultar ou se subscrever em eventos relacionados a esse mesmo tipo de informação contextual.

Devido a essa necessidade, durante este trabalho, foi definido um esquema hierárquico, inspirado na Management Information Base (MIB)[1], para a geração de chaves únicas, chamadas de Context Keys (CK), que identificam possíveis tipos de informação contextual. Usando esse esquema, uma informação contextual é referenciada através de uma sequência de nomes separados por pontos. Por exemplo, uma chave "context.device.location" é utilizada para designar informações contextuais de localização do dispositivo. Assim, qualquer `CAC` que publique esse tipo de informação contextual deve utilizar essa CK. A aplicação que tiver interesse

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

nessa informação deve buscá-la utilizando a mesma CK.

A Figura 4.3 apresenta um trecho já definido da hierarquia para compor uma CK. Quando novos `CACs`, provendo novos tipos de informações contextuais, são desenvolvidos, a hierarquia pode ser estendida a partir de qualquer nó. Por exemplo, caso um novo `CAC` fosse criado para disponibilizar a religião do usuário, a hierarquia poderia ser estendida através da adição de um nó "religion" abaixo do nó "user" para possibilitar o uso da CK "context.user.religion".

Além da necessidade do uso de uma CK para identificar o tipo de contexto, também é necessário especificar como essas informações devem ser representadas. Baseado no metamodelo de contexto proposto por Vieira et al. (2011), foi definido que informações contextuais precisam conter pelo menos quatro campos:

- **ContextKey**: Este campo contém a CK que corresponde ao tipo de informação contextual representada. Por meio dele, aplicações podem verificar quais informações correspondem aos seus interesses em contexto.
- **Source**: Informa a fonte da informação. Informações contextuais podem ser originadas por `CACs` que encapsulam sensores reais, virtuais e lógicos.
- **Values**: Esse campo contem os valores da informação contextual em si. Por exemplo, em uma informação contextual representando a temperatura ambiente em Celsius, esse campo pode ter um valor 28.
- **Timestamp**: Esse campo contém o instante, em milissegundos, em que a informação contextual foi sensoreada.

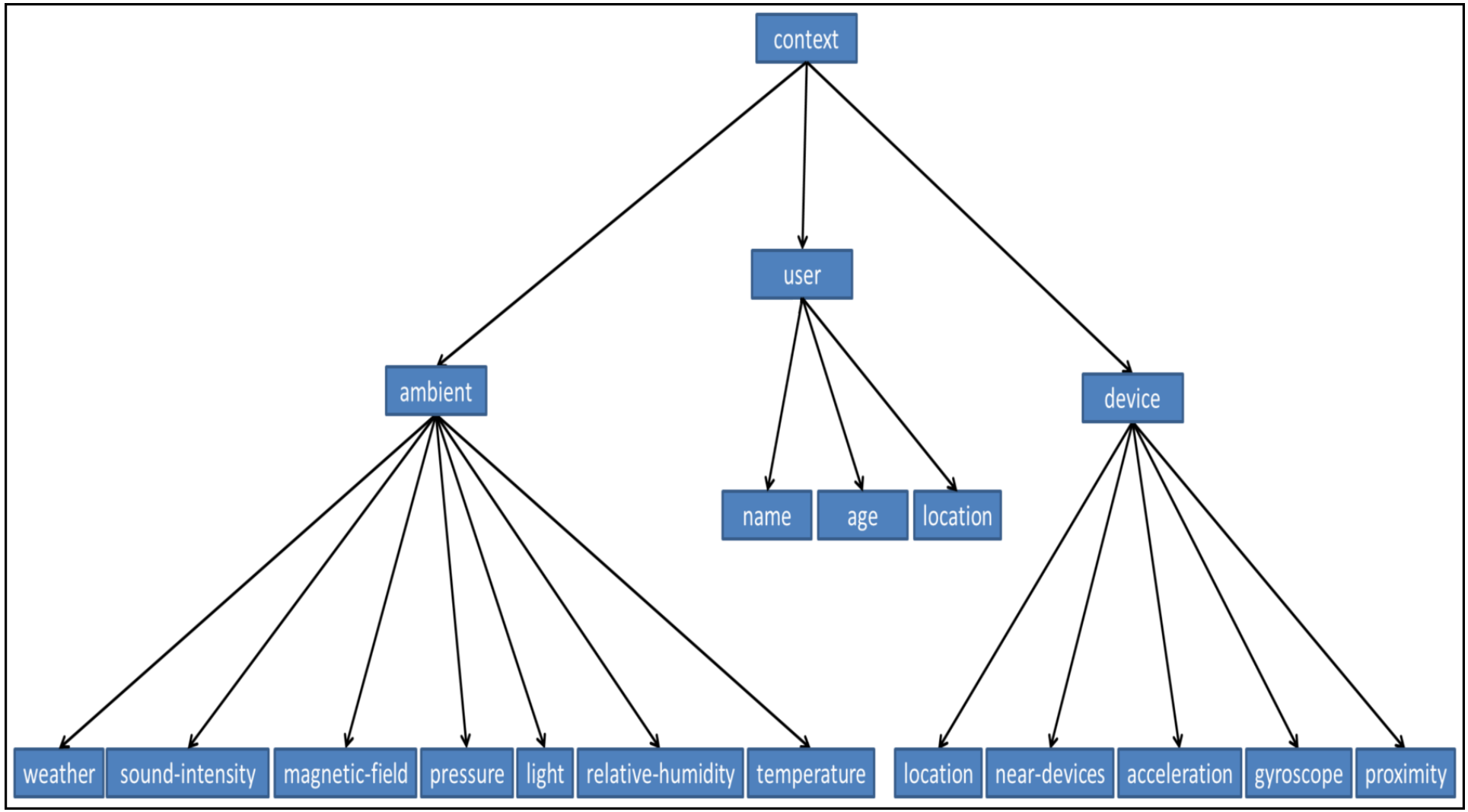

Figura 4.3: Trecho já definido da hierarquia para compor uma CK.

Outros campos também podem ser adicionados de acordo com a necessidade das aplicações ou da informação contextual em si. Em alguns casos, um campo chamado *Accuracy* é adicionado para informar a precisão de uma leitura. Por exemplo, a localização do usuário pode ter uma precisão de 5 metros. Nesse caso o campo *Accuracy* teria o valor 5. A Figura 4.4 mostra exemplos de representação de informações contextuais de acordo com o modelo definido. Esse modelo deve ser seguido por todo e qualquer `CAC` para publicar informações contextuais sensoreadas.

ContextKey	Source	Values	Timestamp	Accuracy
context.device.location	Physical Sensor	38.68551, -9.103271	1337821714638	5
context.ambient.temperature	Physical Sensor	28	1337829018765	3

ContextKey	Source	Values	Timestamp
context.device.owner	LogicalSensor	"John"	1337829056475
context.device.near-devices	LogicalSensor	"Billy", "Jin"	1337829018721
context.user.activity	LogicalSensor	"running"	1337829018721

Figura 4.4: Exemplos de representação de informações contextuais de acordo com o modelo hierárquico proposto.

O modelo de representação de contexto proposto nesse trabalho busca ser simples e se assemelha aos modelos de contexto implementados como chave-valor, apresentados na Seção 2.2.1. Todavia, para lidar com a limitação de expressividade desse tipo de modelos, o modelo proposto liga uma chave a uma estrutura semelhante a uma tupla de banco de dados, que permite vários campos para cada informação, ao invés de um simples valor primitivo (e.g., string e int). As principais vantagens do modelo contextual proposto são a possibilidade de extensão para a criação de novas CKs e também para criação de novos campos para a estrutura que representa a informação contextual (e.g., campo *Accuracy*). Além disso, sua simplicidade permite que novos modelos sejam construídos sobre ele, caso uma aplicação necessite. Por fim, o fato da comunicação entre aplicações e o *framework* acontecer por meio do modelo, cria um desacoplamento entre os seus códigos.

Entre as principais desvantagens do modelo proposto estão o fato de que, também devido a sua simplicidade, relacionamentos de dependências entre contextos não podem ser

apropriadamente modelados e validações formais da modelagem também são dificilmente aplicáveis.

4.3 Componentes de Aquisição de Contexto

A unidade básica de composição da arquitetura do *framework* é o componente de aquisição de contexto (CAC). Toda informação contextual que ele publica é adquirida por componentes desse tipo, retirando assim essa responsabilidade das aplicações. O modelo de componentes de um CAC foi concebido tendo como princípio básico o fácil gerenciamento de ciclo de vida desses componentes por outros componentes externos, característica importante para a realização da adaptação composicional.

A Figura 4.5 apresenta as interfaces requeridas e oferecidas de um CAC, definidas durante o desenvolvimento do *framework*. As interfaces *start* e *stop* permitem que o *framework* possa iniciar e parar o funcionamento do componente quando assim desejar. A interface *isRunning* permite saber se um CAC está em execução. A interface *setProperty* recebe como entrada uma chave e um valor para configurar o comportamento interno do componente, enquanto *getPropertiesKeys* retorna todas as chaves de propriedades configuráveis do CAC. A interface *publish* é requerida por CACs, pois através dela informações contextuais adquiridas são publicadas.

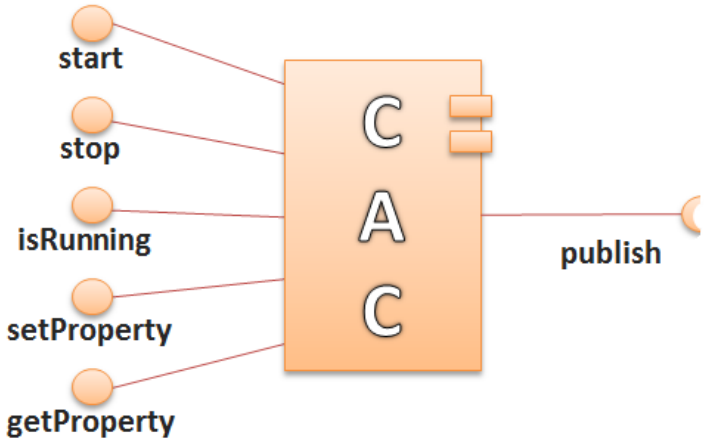

Figura 4.5: Interfaces de um CAC.

Conforme mostra a Figura 4.6, o ciclo de vida de um CAC é composto por apenas 3 estados: *desinstalado*, *instalado* e *ativo*. Quando um CAC é adicionado ao repositório local do dispositivo ele se encontra no estado padrão, que é *desinstalado*. A partir daí ele pode passar para o estado *instalado*. Quando *instalado*, é possível que ele seja iniciado por meio

do método *start* e vá para o estado *ativo* ou volte para o estado *desinstalado*. Enquanto um CAC se encontra *ativo*, sua aquisição de contexto está em execução e informações contextuais estão sendo publicadas. Se o método *stop* for invocado, sua execução para e seu estado volta para apenas *instalado*. Devido ao suporte à adaptação composicional dinâmica e à implantação dinâmica, todas as mudanças do ciclo de vida de um CAC podem ser realizadas com o *framework* instanciado e em execução.

Figura 4.6: Ciclo de vida de um CAC.

Internamente, um CAC pode funcionar como um sensor físico, lógico, virtual ou uma combinação dos três. Componentes de aquisição de contexto que agem como sensores físicos são os que adquirem informações diretamente de sensores fornecidos pelo DM. Também é possível a utilização de informações geradas por outros CACs para realizar inferências e gerar uma nova informação contextual, nesse caso o CAC atua como um sensor lógico. Por exemplo, um componente pode ser implementado para ser baseado na temperatura e na umidade relativa do ar para fornecer uma aproximação da sensação térmica. Caso um CAC adquira informações contextuais a partir de *softwares* ou serviços, ele é considerado virtual. Um exemplo de CAC que atua como sensor virtual seria um que acessasse um serviço *web* meteorológico para fornecer informações climáticas.

Durante esse trabalho de dissertação foram desenvolvidos vários componentes de aquisição de contexto, todavia, com a evolução do modelo de componentes de um CAC (e.g., adição e remoção de interfaces), eles se tornaram incompatíveis com as últimas versões do

framework. Na versão atual do trabalho, estão implementados 4 CACs. Esses componentes podem ser reutilizados por outros desenvolvedores e oferecem os seguintes tipos de informações contextuais: context.ambient.light, context.ambient.temperature, context.device.location, context.device.acceleration.

4.3.1 Desenvolvimento de um CAC

Apesar da possibilidade de reúso dos CACs já disponibilizados, o *framework* também permite ao desenvolvedor de aplicações criar seus próprios componentes de aquisição de contexto. Essa característica é importante, uma vez que não é possível prever todos os tipos de informações contextuais que venham ser requeridas por aplicações, nem mesmo os novos sensores que serão embarcados em DMs no futuro. A Figura 4.7 apresenta a estrutura interna de um CAC que captura a temperatura ambiente. Os principais arquivos e diretórios presentes nesse componente se repetem na criação de qualquer CAC. Eles são: *src*, *lib* e *MANIFEST.MF*.

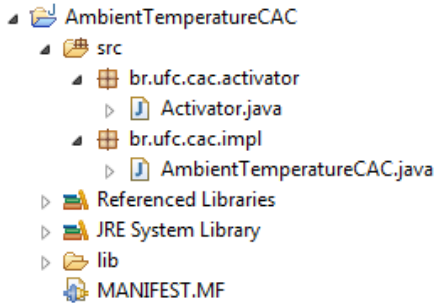

Figura 4.7: Estrutura interna de um CAC.

Dentro de *src* estão presentes todos os códigos fonte, que estão divididos em forma de pacotes. O código fonte do CAC deve ser escrito em Java para manter sua compatibilidade com o *framework*. No exemplo do CAC de temperatura ambiente, é possível visualizar duas únicas classes: Activator e AmbienteTemperatureCAC. A classe Activator é uma classe necessária para todo *bundle* do OSGi. Ela é quem permite ao CAC Manager iniciar e parar o funcionamento do componente. O código do Activator é sempre o mesmo em todo e qualquer CAC. Já a classe AmbienteTemperatureCAC contém o código de aquisição de contexto. Essa classe pode possuir outros nomes e seu código é particular para cada CAC, todavia ela deve sempre implementar uma interface java chamada ISensor, exibida na Figura 4.8. Os métodos de ISensor garantem que todo sensor possuirá as interfaces definidas para um CAC, apresentadas no início da Seção 4.3. O parâmetro Publisher presente no método *start* é a interface utilizada para que o

CAC publique as informações contextuais sensoreadas.

```
public interface ISensor {
    public void start(IAndroidContextProvider platform, Publisher publisher);
    public void stop();
    public void setProperty(String key, String value);
    public String getProperty(String key);
    public String[] getPropertiesKeys();
    public boolean isRunning();
}
```

Figura 4.8: Interface ISensor.

O diretório *lib* contém bibliotecas e APIs utilizadas internamente pelo CAC. Por exemplo, um CAC pode utilizar uma API específica para acessar um sensor. Nesse caso, o arquivo com o binário dessa API é inserido nessa pasta. Por fim, o arquivo *MANIFEST.MF* é onde o desenvolvedor deve declarar metadados sobre o componente. Entre as informações disponíveis no arquivo estão: o tipo de informação contextual oferecida, o nome de quem o construiu, suas dependências, entre outras. Esse arquivo também é uma característica herdada do OSGi. Todo *bundle* OSGi deve possuir um arquivo manifesto. A Figura 4.9 apresenta o *MANIFEST.MF* do CAC de temperatura ambiente. Esse tipo de arquivo é estruturado como um conjunto de chaves e valores.

MANIFEST.MF

```
1 Manifest-Version: 1
2 Built-By: Andre
3 Context-Key: context.ambient.temperature
4 Bundle-Activator: br.ufc.cac.activator.Activator
5 Bundle-ManifestVersion: 2
6 Bundle-Name: br.ufc.cac.ambienttemperature
7 Bundle-SymbolicName: AmbientTemperatureCAC
8 Bundle-Version: 2.0.0
9 Export-Package: br.ufc.cac.activator,br.ufc.cac.impl
10 Import-Package: android.content,android.util,android.hardware,br.ufc.loccam.isensor,br.ufc.loccam.isyssu,br.ufc.loccam.iandroidcc
```

Figura 4.9: Exemplo de arquivo manifesto de um CAC.

Os principais campos presentes no arquivo manifesto são *Context-Key* e *Import-Package*. O campo *Context-Key* é que permite ao *framework* determinar qual CAC oferece que tipo de informação contextual no momento da adaptação dinâmica. Quando novos CACs com novos tipos de informações são criados, a hierarquia de CKs apresentada na Seção 4.2 deve ser estendida para a definir uma CK que corresponda a nova informação contextual. Essa nova CK é então utilizada no campo Context-Key do arquivo manifesto.

O campo *Import-Package* é utilizado para importar dependências que serão utilizados na criação do CAC. Algumas dependências são obrigatórias no manifesto de um CAC.

Por exemplo, a dependência do pacote Java “br.ufc.loccam.isensor” informa que o CAC depende da interface ISensor citada anteriormente. Sem essa dependência o componente não teria acesso à interface ISensor, necessária para a criação do CAC. Por outro lado, a dependência “android.hardware”, por exemplo, só se faz necessária quando o CAC deseja privilégios de acesso aos sensores internos do DM.

Outro campo importante, porém opcional, é o *Interest-Zone*. Nesse campo devem ser inseridas CKs representando os tipos de contexto que são necessários para o funcionamento do CAC. Como o CAC de temperatura ambiente não depende de nenhum tipo de informação contextual provida por outros CACs, esse campo não está presente na Figura 4.9. Ambos os campos *Import-Package* e *Interest-Zone* definem relações de dependências do componente, porém o *Interest-Zone* caracteriza um nível mais alto de abstração: dependência de outros tipos de informação contextual. Assim, qualquer CAC que ofereça tal informação é capaz de satisfazer a dependência.

Uma vez que o componente esteja todo codificado e com o arquivo manifesto configurado, para disponibilizar o CAC é necessário gerar um arquivo JAR que utiliza o MANIFEST.MF como arquivo manifesto e que contenha o binário do projeto. Por fim, para que o CAC possa ser utilizado em um DM com Android, deve-se realizar um segundo processo de compilação. Isso ocorre porque a máquina virtual que roda os códigos escritos em Java no Android é diferente das máquinas virtuais tradicionais. Essa máquina, chamada Dalvik Machine, não roda o mesmo tipo de binário que as máquinas tradicionais. Para realizar essa segunda compilação o Android SDK[2] disponibiliza uma ferramenta chamada *dx*. A Figura 4.10 mostra um exemplo de como realizar a segunda compilação do JAR utilizando essa ferramenta. Após esse processo, um arquivo classes.dex será gerado. Esse arquivo deve ser adicionado ao JAR do CAC, que então, pode ser utilizado normalmente em um dispositivo com Android.

Figura 4.10: Geração do arquivo classes.dex.

[2]http://developer.android.com/sdk/

4.4 Adaptação da Aquisição de Contexto

Em busca de atingir um bom gerenciamento de recursos do DM, assim como um bom gerenciamento dos requisitos em informações contextuais das aplicações, o *framework* proposto oferece a possibilidade de adaptação composicional em tempo de implantação (estática) e em tempo de execução (dinâmica). Toda a adaptação da aquisição de contexto é baseada na implantação, instalação, desinstalação, inicialização, ou parada de componentes `CAC`.

A adaptação estática ocorre durante a implantação do *framework* em um DM. Nesse momento, o desenvolvedor é responsável por determinar quais `CACs` serão implantados juntamente com ele. Devido ao fato de todo `CAC` ser um bundle OSGi, para implantá-lo é necessário copiar o seu arquivo JAR em um diretório dentro do dispositivo. Em tempo de execução, esse diretório será utilizado para buscar os `CACs` disponíveis para instalação, caso seja necessário. Essa adaptação permite que desenvolvedores implantem apenas `CACs` que conhecidamente serão utilizados e que são compatíveis com o *hardware* e o *software* do dispositivo alvo, evitando assim problemas da política do tudo-ou-nada. Por exemplo, ao implantar o *framework* em um dispositivo que não possua GPS, `CACs` que utilizem tal sensor podem não ser incluídos.

A adaptação dinâmica dos `CACs` é baseada na definição de contexto proposta por Viana (2010) e adotada nesse trabalho. Nessa definição, o contexto do sistema pode ser considerado como a interseção entre a ZI (zona de Interesse) e ZO (Zona de observação). No desenvolvimento do *framework*, a ZO foi modelada como o conjunto de todos os `CACs` que estão instalados e executando no sistema em um dado momento. Já a ZI foi modelada como o conjunto de todos os interesses em informações contextuais que estão publicados no sistema, incluindo interesses dos próprios `CACs`. A adaptação do *framework* consiste na alteração de sua ZO, que ocorre por meio da inicialização ou parada de execução de componentes `CAC`.

O objetivo maior da adaptação é alcançar um estado onde a ZO esteja contida na ZI embarcando o máximo de elementos da ZI possível, como mostra a Figura 4.11. Para isso, todos os componentes `CAC` desnecessários para a ZI em um dado instante devem ter sua execução parada. Caso o conjunto que representa a ZI adquira novos elementos, cabe ao mecanismo de adaptação detectar a mudança e buscar componentes, no diretório onde os JARs de `CACs` estão disponíveis, que atendam aos requisitos da ZI. Através dessa adaptação, a aquisição de contexto passa a ter um comportamento elástico onde os recursos do dispositivo são alocados

ou liberados conforme a demanda.

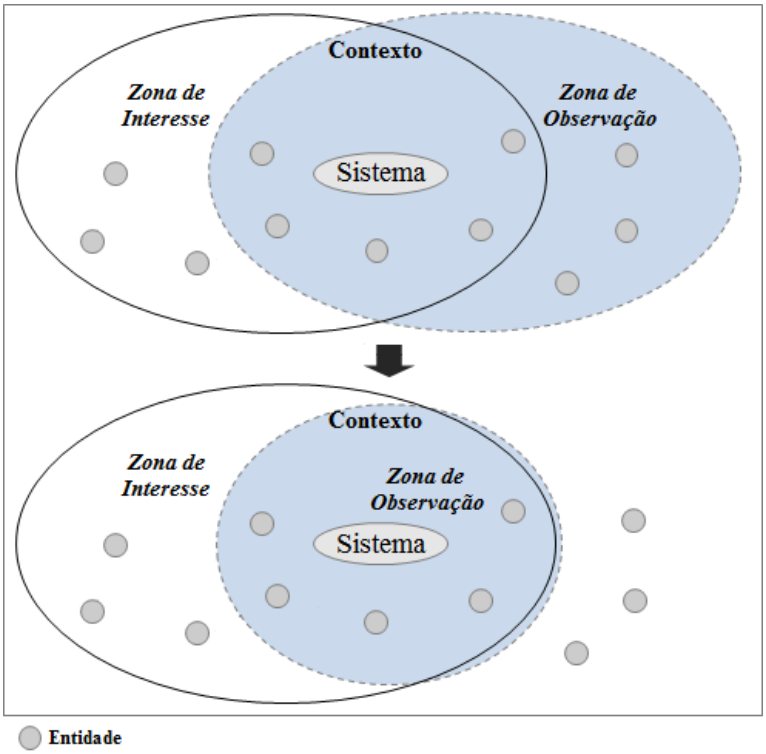

Figura 4.11: Ilustração da adaptação da aquisição de contexto.

4.4.1 Algoritmo de Adaptação Dinâmica

Na implementação do algoritmo de adaptação dinâmica proposto, aplicações, `CACs` e interesses em informações contextuais foram modelados como um grafo direcionado ou orientado (CORMEN et al., 2009). Essa escolha foi feita devido ao fato de um grafo ser uma representação natural de arquiteturas de sistemas (BRADBURY et al., 2004). No grafo modelado, aplicações e `CACs` são considerados vértices enquanto seus interesses são representados como arestas direcionadas. A origem das arestas indicam quem possui o interesse e o destino da aresta representa quem o supre, como mostra a Figura 4.12.

Toda vez que um interesse é adicionado ao *framework*, o algoritmo procura um vértice que represente um `CAC` capaz de fornecer tal informação contextual. Caso o vértice já exista, uma aresta entre o vértice que representa quem possui o interesse e o `CAC` encontrado é criada. A aresta, nesse caso, representa um interesse satisfeito pelo vértice que representa o `CAC`. Quando o vértice já existe, isso significa que outra aplicação ou `CAC` em execução já publicou interesse pelo tipo de contexto provido por ele. No caso do vértice não existir, tenta-se

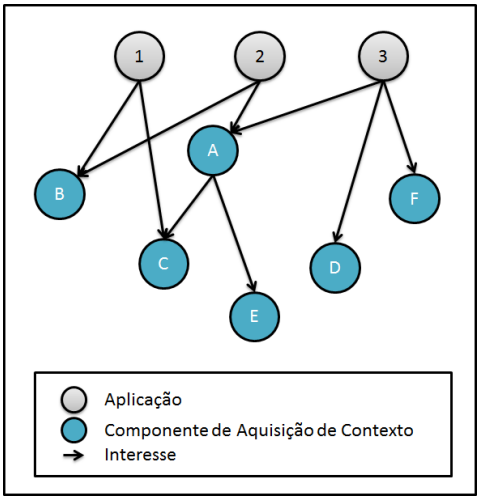

Figura 4.12: Ilustração do grafo do algoritmo.

localizar um CAC no repositório local do dispositivo que satisfaça o interesse publicado. Esse CAC é, então, inserido no grafo e todos os seus interesses são publicados para o algoritmo, que repete recursivamente todo o processo de quando um novo interesse é publicado. Caso os interesses do CAC não sejam todos satisfeitos, ele é removido do grafo juntamente com todos os seus interesses. Então, outro CAC do repositório que satisfaça o interesse inicial é adicionado ao grafo, repetindo novamente o processo.

Para calcular a complexidade desse trecho do algoritmo, é considerado como o pior dos casos um estado do grafo onde existe apenas um vértice de aplicação no grafo e n componentes disponíveis no repositório para instalação. Além disso, todos os n componentes do repositório possuem interesse em todos os outros. Nesse caso, quando uma aplicação publicar algum interesse que qualquer componente do repositório satisfaça, será construído um grafo completo (CORMEN et al., 2009) com n vértices (componentes), que possui $\frac{n(n-1)}{2}$ arestas (interesse de todos os componentes). Sendo assim, a complexidade para construir tal grafo e, consequentemente, de inserção de um novo interesse em informação contextual é $O(n^2)$.

A outra parte importante do algoritmo se refere a retirada de um interesse por parte da aplicação ou CAC. Quando uma aplicação informa ao *framework* que não possui mais interesse em uma informação contextual, o algoritmo procura pela aresta correspondente no grafo. Ao encontrar a aresta, ele a remove e verifica se o CAC que satisfazia o interesse ainda é interesse de outra aplicação ou CAC. Caso ninguém mais possua interesse no CAC, ele é removido. Quando ele é removido todos os seus interesses também são removidos, disparando todo o processo novamente para cada interesse. A Figura 4.13 apresenta dois diagramas com os processos

de adaptação na adição e na remoção de um interesse.

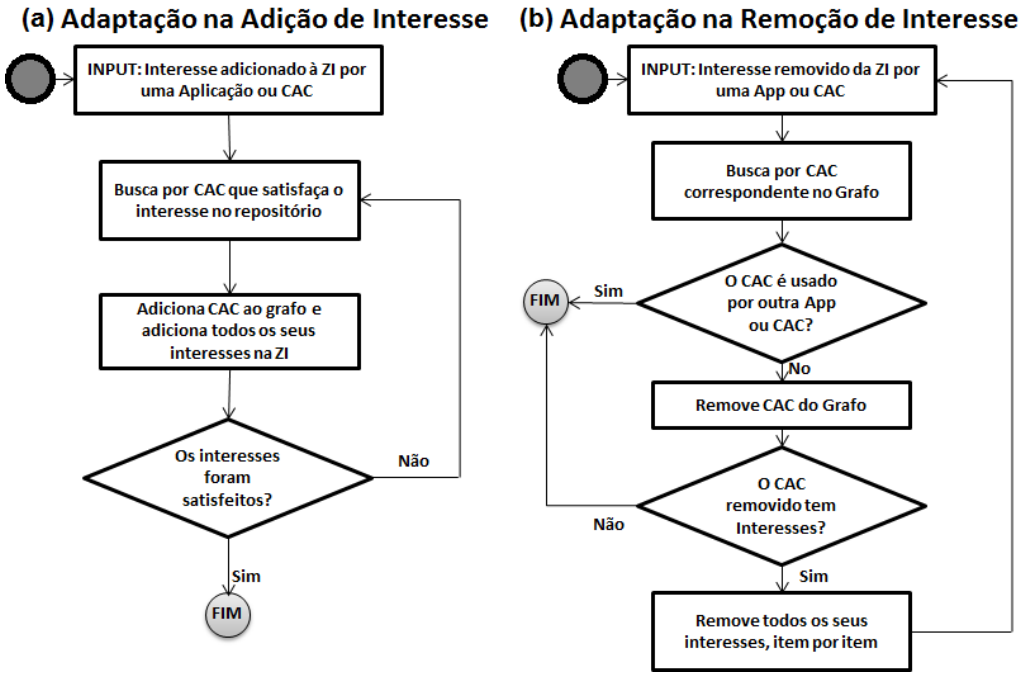

Figura 4.13: Processo de adaptação na (a) adição e (b) remoção de interesse.

Para calcular a complexidade desse trecho do algoritmo também é levado em consideração o pior dos casos. Esse cenário ocorre quando a remoção de um único interesse resulta na remoção de todos os vértices do grafo, através da remoção de todas as arestas. Um exemplo de tal cenário ocorre quando no grafo existe apenas uma aplicação que possui interesse satisfeito por um componente que depende de todos os outros, como mostra a Figura 4.14. A complexidade resultante seria então a da remoção de todas as m arestas, e consequentemente dos vértices, que é de $O(m)$.

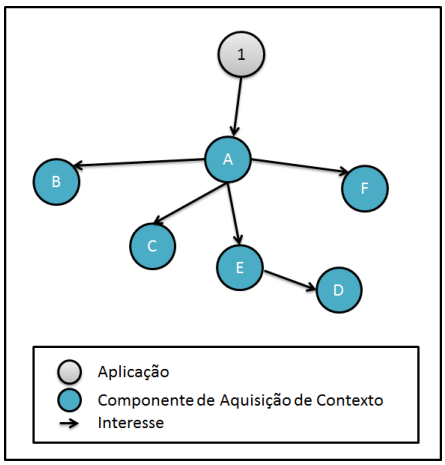

Figura 4.14: Grafo de exemplo de pior dos casos para uma retirada de interesse.

4.5 LoCCAM

O LoCCAM (MAIA et al., 2013) é uma infraestrutura de suporte a sistemas sensíveis ao contexto projetada por um grupo de pesquisadores do GREat[3]. Esse *middleware* considera como principal fonte de informações contextuais os sensores físicos, lógicos e virtuais presentes no próprio dispositivo móvel em que ele se encontra.

Apesar da possibilidade do *framework* ser utilizado diretamente pelas aplicações, algumas funcionalidades importantes em infraestruturas de suporte a sistemas sensíveis ao contexto não estão presentes nele. Comparando à arquitetura conceitual de infraestruturas de Baldauf et al. (2007), o *framework* proposto é equivalente as camadas de Aquisição de Informações de Baixo Nível e de Preprocessamento. Assim, apesar de fornecer serviços de aquisição e enriquecimento de informações contextuais, ele não disponibiliza meios sofisticados para acesso a essas informações adquiridas e nenhum sistema de armazenamento das mesmas. Buscando suprir essa deficiência, o *framework* proposto foi inserido no *middleware* LoCCAM (Loosely Coupled Context Acquisition Middleware). Como consequência dessa inserção, outras contribuições foram realizadas durante este trabalho de dissertação para integrar o funcionamento dele ao LoCCAM.

O desenvolvimento do LoCCAM foi baseado em cinco princípios:

1. Desenvolvimento de requisitos de aquisição de contexto seguindo o princípio da CBSE;
2. Possibilidade de adaptação da aquisição de contexto para economia de recursos;
3. Disponibilidade de informações contextuais através de consultas síncronas e notificações assíncronas por meio de subscrição;
4. Desacoplamento entre as aplicações e o código de aquisição de contexto; e
5. Transparência na aquisição de informação contextual.

Muitas desses princípios citados são alcançados por meio da simples utilização do *framework* proposto. Os outros princípios foram obtidos por meio da utilização da ferramenta SysSU, apresentada na Seção 3.1.5, ou da junção das duas. Por exemplo, o desenvolvimento

[3]O grupo é composto pelos integrantes Rossana Andrade, Windson Viana, André Fonteles, Benedito Neto, Márcio Maia e Rômulo Gadelha.

da aquisição de contexto seguindo o princípio da CBSE, assim como a sua adaptação, são características herdadas diretamente do *framework* proposto. Já a disponibilidade de acesso as informações de forma síncrona ou assíncrona é uma característica obtida por meio da utilização do SysSU. O desacoplamento entre as aplicações e o *middleware* é uma característica oriunda da junção das duas ferramentas.

A Figura 4.15 mostra a arquitetura do LoCCAM e suas principais entidades: o *framework* proposto CAM e a ferramenta SysSU. O LoCCAM funciona como uma instância única embarcada no dispositivo móvel e acessível por várias aplicações. Toda a comunicação entre o *middleware* e a aplicação ocorrem por meio do SysSU. É nele que as aplicações publicam quais interesses em informações contextuais o LoCCAM deve suprir (zona de interesse). Sempre que um novo interesse é adicionado, o SysSU notifica o `Adaptation Reasoner` sobre a mudança, o que pode disparar um processo de adaptação da estrutura dos `CACs`. Também é no SysSU que

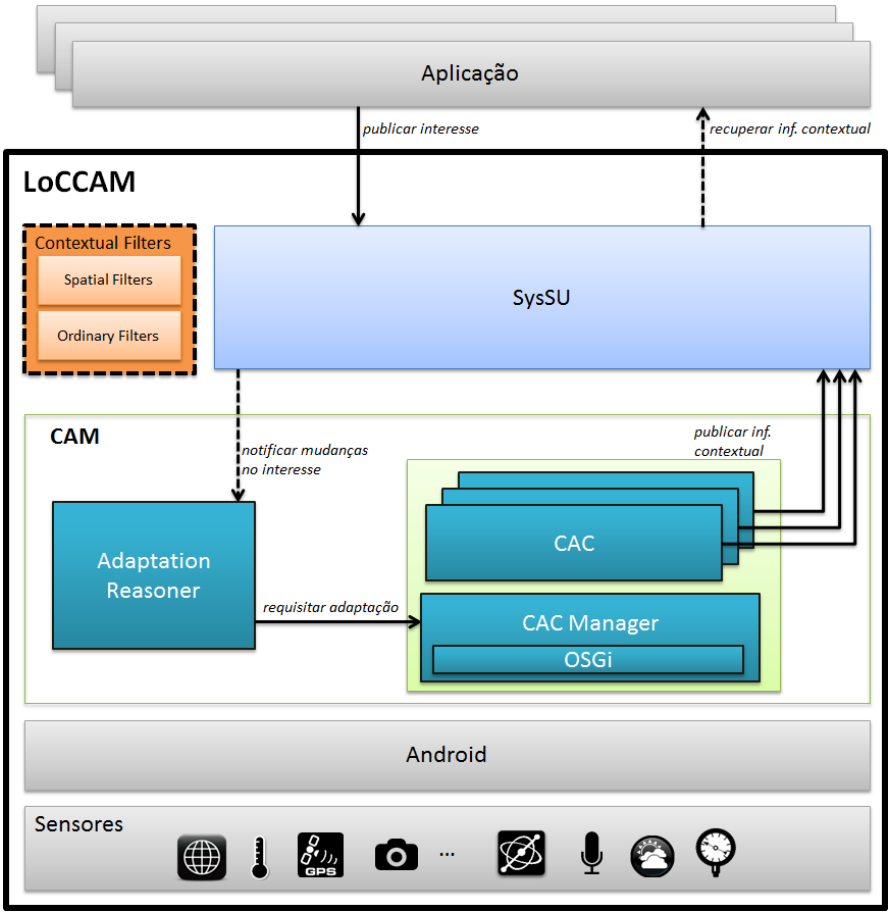

Figura 4.15: Arquitetura do LoCCAM.

toda informação contextual sensoreada pelos `CACs` é publicada. Essas informações são então disponibilizadas para consultas síncronas ou por meio de eventos assíncronos para aplicações

subscritas. Para acessar as informações contextuais, aplicações podem fazer uso de classes chamadas filtros contextuais. Tais classes podem ser reutilizadas ou definidas pelos próprios desenvolvedores de aplicações. Os filtros contextuais são explicados em detalhes na próxima seção.

4.6 Acesso a Informações Contextuais no LoCCAM

Como citado no início da Seção 4.5, toda a comunicação entre aplicações e o *middleware* LoCCAM ocorre por meio do SysSU. É nele que todo `CAC` publica informações contextuais sensoreadas. Também é através dele que aplicações podem realizar consultas por tais informações ou serem notificadas automaticamente caso estejam subscritas.

Originalmente, o funcionamento do SysSU era distribuído e fazia uso de JavaScript como linguagem para implementação dos filtros usados para acessar dados. Nesse trabalho de dissertação, a arquitetura do SysSU foi adaptada para ser completamente embarcada em um único dispositivo móvel e ser utilizada no LoCCAM. Além disso, devido a restrições de desempenho na execução de filtros implementados em JavaScript dentro de um DM, a linguagem para criação dos mesmos também foi alterada para Java.

4.6.1 Consultas

Através do uso de *templates* e filtros do SysSU, o LoCCAM permite dois tipos distintos de consultas. A primeira delas é a que busca qualquer informação contextual de um tipo, independente de seu valor. Esse tipo de busca é realizada fazendo uso apenas de *templates*. Por exemplo, caso uma aplicação esteja em busca da temperatura ambiente onde o dispositivo se encontra, ela pode fazer uso do *template* {(ContextKey,“context.ambient.temperature”)}. Esse *template* retorna todas as tuplas do SysSU que contiverem um campo chamado *ContextKey* juntamente com o valor “context.ambient.temperature”. Da mesma forma, outras CKs devem ser utilizadas para buscar informações contextuais de outros tipos.

Por outro lado, para realizar consultas ou subscrições com critérios mais sofisticados, um filtro deve ser utilizado juntamente ao *template*. Por exemplo, no caso anterior, onde a informação requerida é a temperatura ambiente, caso a aplicação esteja em busca apenas de tuplas que representem temperaturas acima de 30 graus, um filtro como o da Figura 4.16 pode

ser utilizado juntamente ao *template* anterior.

```
public boolean filter(Tuple tuple) {
    if(constainsContextKey(tuple, contextKey)) {
        for (int i = 0; i < tuple.size(); i++) {
            if(tuple.getField(i).getName().equals("Values")) {

                List<Double> doubleValues = (List<Double>)tuple.getField(i).getValue();
                double tupleValue = doubleValues.get(valueIndex);

                if(tupleValue > 30)
                    return true;
                else
                    return false;
            }
        }
    }
    return false;
}
```

Figura 4.16: Exemplo de filtro contextual.

Caso necessário, filtros mais complexos podem ser desenvolvidos realizando consultas internas. Por exemplo, no filtro da temperatura, além de verificar se a temperatura está acima de 30 graus, poderia ser realizada uma consulta, dentro do método *filter*, com a "CK context.ambient.light", para avaliar se além de quente o dia também está ensolarado. Tais filtros podem ser utilizados por aplicações para realizar operações sofisticadas como propor que o usuário vá a uma praia ou a uma sorveteria próxima. Apesar da versatilidade dos filtros complexos, realizar uma consulta dentro de um filtro é potencialmente custoso para os recursos do dispositivo, pois o código de um filtro pode ser executado várias vezes em uma única consulta ao SysSU.

4.6.2 Filtros Espaciais

A Localização do usuário é uma informação contextual que desempenha papel importante em aplicações sensíveis ao contexto (BETTINI et al., 2010). Várias aplicações, como por exemplo em (VIANA et al., 2007) e (BRAGA et al., 2012), consideram localização como a principal informação contextual, a qual outras informações contextuais são então atreladas. Todavia, analisando *surveys* de *frameworks* e *middlewares* de suporte a sensibilidade ao contexto como (BALDAUF et al., 2007), (KELING et al., 2012) e (HONG et al., 2009), é possível perceber que essas infraestruturas oferecem poucos serviços ou mecanismos para lidar com informações espaciais.

No LoCCAM é possível estender filtros do SysSU ou mesmo reutilizar filtros já definidos para gerenciar informações geográficas. Como parte do trabalho dessa dissertação, foram desenvolvidos dois filtros reutilizáveis para a realização de consultas síncronas e subscrições baseada em informações espaciais: DistanceFilter e PolygonFilter.

O primeiro filtro, chamado DistanceFilter, é utilizado quando uma aplicação deseja saber se/quando o usuário está dentro de uma distância (raio) mínima de uma coordenada geográfica. Uma aplicação pode, por exemplo, utilizando esse filtro em uma subscrição no LoCCAM, ser notificada quando o usuário estiver a menos de 100 metros de um ponto turístico. O DistanceFilter foi desenvolvido utilizando uma classe chamada Location[4] presente no Android Framework. A classe Location possui um método chamado *distanceTo* que retorna a distância aproximada entre duas coordenadas geográficas em metros.

Ao criar um objeto do tipo DistanceFilter, é necessário especificar uma localização geográfica e um raio. Esses dados serão utilizados para realizar comparações dentro do método *filter* do filtro. A Figura 4.17 apresenta a implementação desse método em DistanceFilter e um exemplo visual de como ele funciona. O método *filter* recebe como parâmetro uma tupla com uma localização geográfica. Nas primeiras linhas de código, essa informação é extraída do campo "Value" da tupla. Esses dados são então comparados e o método retorna *true* se a distância entre as duas localizações é menor que o raio escolhido. Quando o método retorna trupe, significa que aquela tupla "passou" pelo filtro.

```
public boolean filter(Tuple tuple) {
    for (int i = 0; i < tuple.size(); i++) {
        if(tuple.getField(i).getName().equals("Values")) {
            List<String> stringValues = (List<String>)tuple.getField(i).getValue();
            List<Double> values = new ArrayList<Double>();

            for (String value : stringValues)
                values.add(Double.parseDouble(value));

            userLocation = new Location("");
            userLocation.setLatitude(values.get(0));
            userLocation.setLongitude(values.get(1));
        }
    }

    return location.distanceTo(userLocation) <= distance;
}
```

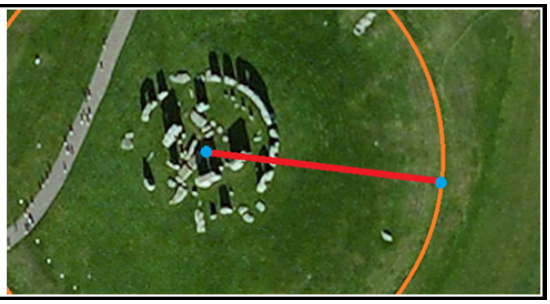

Figura 4.17: Implementação do método filter de DistanceFilter e exemplo visual de um raio que pode ser definido.

O segundo filtro, chamado PolygonFilter, é usado para determinar se um usuário está dentro de uma ou mais áreas geográficas determinadas por um conjunto de coordenadas. A principal diferença entre DistanceFiltere e PolygonFilter é que o segundo permite a definição de polígonos complexos, como mostra a Figura 4.18, enquanto o primeiro, por ser baseado em

[4]http://developer.android.com/reference/android/location/Location.html

distância (raio), apenas de circunferências. Na implementação desse filtro também foi usada uma API de terceiros, dessa vez chamada JST[5]. Essa API foi utilizada para testar interseções entre os polígonos e o ponto da localização do usuário.

```
public boolean filter(Tuple tuple) {

    for (int i = 0; i < tuple.size(); i++) {

        if(tuple.getField(i).getName().equals("Values")) {
            @SuppressWarnings("unchecked")
            List<String> stringValues = (List<String>)tuple.getField(i).getValue();

            List<Double> values = new ArrayList<Double>();

            for (String value : stringValues) {
                values.add(Double.parseDouble(value));
            }

            userCoordinate = new Coordinate(values.get(0), values.get(1));
        }
    }

    Point userPoint = geometryFactory.createPoint(userCoordinate);
    boolean result = false;

    for (Polygon polygon : placesPolygon) {
        if(polygon.intersects(userPoint)) {
            result = true;
            break;
        }
    }

    return result;
}
```

Figura 4.18: Implementação do método filter de PolygonFilter e exemplo visual de um polígono que pode ser definido.

Na instanciação de um objeto do tipo PolygonFilter deve ser passado como parâmetro uma lista de polígonos representando áreas geográficas. Assim como o filtro anterior, o método *filter* de PolygonFilter também inicia pegando o valor do campo "Value" da tupla, que contem uma localização geográfica. Em seguida, através do uso da API JST, o método verifica se a localização contida na tupla está dentro de algum dos polígonos de áreas definidas previamente e retorna *true*, em caso afirmativo.

Assim como os dois filtros espaciais criados neste trabalho, é possível que outros desenvolvedores criem novos filtros de acordo com as necessidades específicas de suas aplicações e os distribuam como uma classe Java, para propósitos de reúso.

4.7 Conclusão

Nesse capítulo foi apresentada a proposta de um *framework* de aquisição de contexto adaptável chamado CAM (Context Acquisition Manager). Esse *framework* busca diminuir o problema do tudo-ou-nada e atingir um bom gerenciamento dos recursos do dispositivo móvel através da adaptação estática e dinâmica de sua aquisição de contexto. A unidade básica de

[5]http://www.vividsolutions.com/jts

composição do *framework* proposto é o componente de aquisição de contexto. São esse componentes os responsáveis por adquirir e publicar toda informação contextual. É através da adição, remoção, inicialização e parada desses componentes que ocorre a adaptação estática e dinâmica do *framework*.

Nesse capítulo também foram discutidas como ocorrem as adaptações estática e dinâmica da aquisição de contexto do *framework* proposto. A adaptação estática ocorre por meio da escolha de quais `CACs` serão adicionados na implantação do *framework*, o que permite retirar componentes desnecessários para certos DM e aplicações. Já a adaptação dinâmica se baseia em um algoritmo proposto a partir da definição de contexto de Viana (2010) e modela a ZO e ZI como um grafo contendo aplicações, `CACs` e seus respectivos interesses. A adaptação dinâmica possibilita um bom gerenciamento de recursos e requisitos de aplicações que a utilize, uma vez que novos componentes podem ser iniciados em tempo de execução para suprir novos requisitos, e parados, para economizar recursos.

Também foi apresentado um *middleware* chamado LoCCAM, onde o *framework* CAM foi inserido. LoCCAM foi desenvolvido por um grupo de pesquisadores do GREat e, através do uso do SysSU, permite que as informações publicadas pelo *framework* proposto sejam acessadas síncrona e assincronamente. A inserção do *framework* proposto no LoCCAM bem como todas as adaptações no código necessárias são partes das contribuições deste trabalho de dissertação. Entre as adaptações realizadas estão a refatoração do SysSU para execução em um único dispositivo móvel com Android e a mudança da implementação de seus filtros de JavaScript para Java. Outra contribuição decorrente é a proposta de uma forma de utilização dos *templates* e filtros do SysSU para acessar as informações contextuais geradas. Essa forma permitiu a criação de filtros como os espaciais, que foram implementados durante este trabalho de dissertação. Tais filtros podem ser reusados por aplicações que necessitem realizar operações espaciais como interseção e distância na hora de realizar consultas e subscrições no LoCCAM.

O código fonte do *framework* CAM e dos `CACs` implementados, bem como do LoCCAM estão disponíveis para *download* e contribuições no endereço https://github.com/Andre-Fonteles/LoCCAM/. Futuras versões estarão disponíveis em http://loccam.great.ufc.br .

O próximo capítulo apresenta exemplos de códigos de utilização do *framework* proposto através de uma implementação de uma aplicação como prova de conceito. No próximo capítulo também será avaliado o impacto da adaptação dinâmica nos recursos do dispositivo.

5 PROVA DE CONCEITO E AVALIAÇÃO

Com o intuito de avaliar o *framework* proposto e de apresentar como é possível utilizá-lo, foi implementada uma aplicação como prova de conceito. A implementação da aplicação, chamada Context-Track, demonstra passo a passo como é possível utilizar o LoCCAM, juntamente com o *framework*, para o desenvolvimento de aplicações sensíveis ao contexto e também como reutilizar filtros para realizar subscrições em eventos contextuais. Na aplicação prova de conceito foi utilizado o filtro PolygonFilter, criado durante este trabalho de dissertação, e apresentado na Seção 4.6.2.

Para avaliar a adaptação dinâmica da aquisição de contexto bem como o impacto de sua utilização nos recursos do dispositivo móvel, foi desenvolvido um protótipo de uma segunda aplicação idealizada. O protótipo foi utilizado para realização de testes que avaliaram a economia dos recursos do dispositivo (bateria e CPU) comparando uma abordagem com e sem a adaptação dinâmica.

Na Seção 5.1 é apresentada a prova de conceito e como ela foi implementada passo a passo. Já a Seção 5.2 expõe uma aplicação idealizada, chamada CareOnTheGo, sobre a qual o protótipo foi construído para realização de testes do mecanismo de adaptação dinâmica e resultados obtidos.

5.1 Prova de Conceito

Nessa seção é descrito como uma aplicação sensível ao contexto pode ser implementada utilizando o *middleware* LoCCAM e, consequentemente, também o *framework* proposto. Para tanto, foi definido um cenário onde uma aplicação monitora a trajetória de um usuário. Essa trajetória é composta por um conjunto ordenado de coordenadas por onde o usuário passou e para cada coordenada são atreladas mais informações contextuais. Por exemplo, uma coordenada pode ser enriquecida com a temperatura e o nível de luminosidade do local no momento em que o usuário esteve lá.

O cenário descrito anteriormente, onde existe um tipo de *log* composto por localizações atreladas a informações contextuais é comum em muitas aplicações sensíveis ao contexto, como por exemplo: Photomap (VIANA et al., 2007) e Captain (BRAGA et al., 2012). Photo-

map é um sistema que gera anotações contextuais de uma foto baseada na localização em que ela foi tirada e depois permite ao usuário gerenciar essas fotos em um mapa. Já a aplicação Captain, mais semelhante ao cenário proposto, busca mapear a trajetória de um usuário, através dos sensores de seu *smartphone*, e adicionar informações contextuais em cada posição que faz parte da trajetória para geração automática de uma espécie de diário de bordo na *web*.

Em nosso cenário, além de monitorar a trajetória, foi definida uma região do mapa, que representa uma praça, onde a aplicação é notificada sempre que o usuário está dentro. Tal tipo de notificação é uma característica comumente encontrada em aplicações baseadas em localização e sensíveis ao contexto. Por exemplo, algumas aplicações turísticas apresentam informações, fotos e vídeos quando um usuário está dentro de uma região próxima a um monumento histórico.

De acordo com o cenário descrito foi desenvolvida uma aplicação para Android, batizada Context-Track, utilizando o LoCCAM. A Figura 5.1 mostra uma representação visual de algumas coordenadas da trajetória sensoreadas durante um teste real de uso da aplicação. Nesse teste, o usuário caminhou por alguns quarteirões nas mediações de sua residência. A Figura 5.1 também mostra uma região ao redor de uma praça onde a aplicação se subscreveu para ser notificada sempre que o usuário se encontrasse dentro. Tal subscrição foi realizada pela utilização do filtro PolygonFilter, apresentado na Seção 4.6.2.

Figura 5.1: Representação visual de uma trajetória sensoreada durante um teste com a aplicação Context-Track.

5.1.1 Implementação de Context-Track

Quando uma aplicação deseja utilizar o LoCCAM, o cenário mais comum é de que o *middleware* ainda não esteja implantado no DM. A instalação do LoCCAM é simples. Assim como qualquer aplicativo do Android, ele é distribuído na forma de um arquivo APK que pode ser executado no dispositivo para efetuar sua instalação. Ao contrário da política do tudo-ou-nada adotada por muitas infraestruturas, esse arquivo APK contém apenas os dois componentes indispensáveis que toda instância do LoCCAM necessita para execução: Adaptation Reasoner e `CAC Manager`. Uma vez instalado, o LoCCAM inicia sua execução na forma de um serviço no dispositivo, todavia, ele ainda não é capaz de prover nenhum tipo de contexto para aplicações.

Para que o LoCCAM possa iniciar seu monitoramento do contexto, é necessário disponibilizar um ou mais `CACs` para ele. Isso é feito através da cópia de arquivos JARs para dentro do seu diretório que funciona como repositório (sdcard/felix/availablebundles). Quando um JAR é inserido no repositório, o componente é automaticamente instalado sem a necessidade de parar a execução do *middleware* (implantação dinâmica).

Uma vez que o LoCCAM esteja em funcionamento e com `CACs` instalados, as aplicações podem publicar seus interesses no SysSU para iniciar a aquisição de contexto. A partir daí, o LoCCAM, através do *framework* proposto, inicia os componentes de aquisição de contexto que passam a publicar as informações contextuais no SysSU e as aplicações estarão aptas a realizarem consultas síncronas ou se subscreverem por informações contextuais. Como dito anteriormente, toda essa comunicação entre as aplicações e o *middleware* ocorre por meio do SysSU, que é disponibilizado como um serviço do Android[1].

No caso da aplicação Context-Track, além do interesse na localização do dispositivo do usuário, a aplicação também utiliza a temperatura, umidade relativa e luminosidade do ambiente. A Figura 5.2 apresenta o trecho de código que publica o interesse nessas informações no SysSU. Dois parâmetros são necessários para a publicação de um interesse. O primeiro é o nome da aplicação, que servirá como um identificador para o LoCCAM. O segundo é o interesse em si, expressado por meio de uma CK.

O próximo passo realizado na implementação de Context-Track foi fazer uma subscrição, de forma que a aplicação fosse notificada pelo SysSU sempre que uma nova informação

[1] http://developer.android.com/guide/components/aidl.html

```
sysSUAndroid.publishInterests("appName", "context.device.location");
sysSUAndroid.publishInterests("appName", "context.ambient.light");
sysSUAndroid.publishInterests("appName", "context.ambient.temperature");
sysSUAndroid.publishInterests("appName", "context.ambient.relative-humidity");
```

Figura 5.2: Trecho de código da publicação de interesses da aplicação Context-Track.

contextual de localização do usuário fosse publicada por um CAC. Para se subscrever no SysSU, é necessário utilizar uma classe que implemente uma interface do SysSU chamada IReaction. Uma *reaction*, classe que implementa tal interface, é utilizada no momento de subscrição e possui um método, chamado *react*, que é invocado quando uma tupla publicada no SysSU está dentro dos critérios da subscrição. Alem desse método, existem também os *getTemplate* e *getFilter*, que são responsáveis, respectivamente, por retornar uma instância do *template* e do *filtro* que serão usados na subscrição. Caso a *reaction* não utilize um dos dois, o método pode retornar nulo como resposta.

A Figura 5.3 apresenta os principais trechos de código de uma *reaction* criada durante a implementação de Context-Track. Essa *reaction*, chamada LocationReaction, é utilizada para que a aplicação seja notificada sobre cada nova informação de localização publicada no SysSU. Para tanto, o *template* utilizado segue o padrão {(ContextKey,“context.device.location”)} e não foi definido nenhum filtro.

```
class LocationReaction implements IReaction {

   public Template getTemplate() {
       return (Template)new Template().addField("ContextKey", "context.device.location");
   }

   public IFilter getFilter() {
       return null;
   }

   public void react(Tuple locationTuple) {
      List<Tuple> lightTuple = sysSUAndroid.read("context.ambient.light");
      List<Tuple> temperatureTuple = sysSUAndroid.read("context.ambient.temperature");
      List<Tuple> humidityTuple = sysSUAndroid.read("context.ambient.relative-humidity");

      ...

      persist(locationTuple, lightTuple, temperatureTuple, humidityTuple);
   }
   ...
}
```

Figura 5.3: Trecho resumido de código da LocationReaction da aplicação Context-Track.

O método *react* de LocationReaction, é quem receberá cada nova tupla contextual

de localização do usuário sensoreada pelo LoCCAM. Sempre que ele é invocado, é efetuada uma leitura síncrona das demais informações contextuais que serão associadas a localização: luz, temperatura e umidade relativa do ambiente. Na implementação de Context-Track, essa leitura é feita através do método *read* onde é passado como parâmetro apenas a CK que representa o tipo de contexto requisitado, como visto na Figura 5.4. Apesar disso, também é possível realizar leituras com critérios mais complexos utilizando *templates* e filtros.

Além da implementação da classe LocationReaction, também foi implementada uma segunda *reaction*, chamada PolygonReaction. Essa outra é utilizada para realizar uma subscrição que alerte o aplicativo sempre que o usuário entrar na área da praça, demarcada por um quadrilátero na parte esquerda da Figura 5.1. O *template* utilizado em PolygonReaction é o mesmo utilizado em LocationReaction, pois ambas se referem a informações de localização, todavia, dessa vez, um filtro é utilizado. A Figura 5.4 apresenta uma versão, também resumida, de PolygonReaction. Nela é possível observar o reúso do filtro PolygonFilter. Para instanciação do PolygonFilter foram utilizadas as coordenadas que definem a área da praça, como pode ser visto no método *getTemplate*.

```
class PolygonReaction implements IReaction {

   public Template getTemplate() {
       return (Template)new Template().addField("ContextKey", "context.device.location");
   }

   public IFilter getFilter() {
       Coordinate[] squareCoordinates = new Coordinate[5];
       squareCoordinates[0] = new Coordinate(-3.730942, -38.518051);
       squareCoordinates[1] = new Coordinate(-3.730674, -38.517778);
       squareCoordinates[2] = new Coordinate(-3.73084, -38.517225);
       squareCoordinates[3] = new Coordinate(-3.731204, -38.517332);
       squareCoordinates[4] = new Coordinate(-3.730942, -38.518051);
       PlaceFilter placeFilter = new PlaceFilter(squareCoordinates);

       return placeFilter;
   }

   public void react(Tuple tuple) {
       doSomething(tuple);
   }
}
```

Figura 5.4: Trecho resumido de código da PolygonReaction da aplicação Context-Track.

Por fim, as *reactions* criadas são utilizadas para subscrição no LoCCAM, onde o SysSU ficará responsável por chamar os seus métodos *react* sempre que uma nova informação

contextual atender o critério de estabelecido por elas. A Figura 5.5 apresenta o trecho de código da subscrição das duas *reactions* criadas em Context-Track.

```
sysSUAndroid.subscribe(new LocationReaction(), "put", null);
sysSUAndroid.subscribe(new PolygonReaction(), "put", null);
```

Figura 5.5: Trecho de código da subscrição feita por Context-Track.

Essa prova de conceito mostra como o *framework* de aquisição de contexto CAM e o *middleware* LoCCAM são desacoplados do código da aplicação. Na criação de uma aplicação móvel, um desenvolvedor deve se preocupar apenas com a criação de filtros, quando necessário, e realização de consultas síncronas e assíncronas de informações contextuais. Nenhum código de acesso a sensores físicos, lógicos ou virtuais é necessário para a implementação de uma aplicação. Essa característica é que permite ao *framework* proposto adaptar em tempo de execução a forma como o contexto é adquirido sem que a aplicação se preocupe com as mudanças. Além disso, o desenvolvimento de aplicações sensíveis ao contexto também é facilitado, uma vez que a complexidade da aquisição de contexto está encapsulada no *framework*.

A implementação da aplicação Context-Track também mostra como é possível praticar o reúso de filtros já implementados previamente. O reúso de filtros também permite que a complexidade no manuseio de certas informações contextuais seja delegada a terceiros, como por exemplo no caso do filtro espacial PolygonFilter, que lida com informações geoespaciais.

5.2 Avaliação do Impacto da Adaptação Dinâmica

Além do desenvolvimento da aplicação Context-Track, também foi idealizada uma aplicação móvel e sensível ao contexto, chamada CareOnTheGo, para avaliação do algoritmo da adaptação dinâmica composicional e do seu impacto na utilização de recursos do dispositivo. Sobre os requisitos dessa aplicação, foi desenvolvido um protótipo que possibilitasse a realização de testes com o mecanismo de adaptação.

A aplicação CareOnTheGo foi idealizada para prover dicas de comportamentos saudáveis baseado nas condições do ambiente e no local em que o usuário se encontra: casa, trabalho ou na rua. No protótipo desenvolvido esses locais foram modelados respectivamente como: *home*, *work* e *outdoor*. Para obter informação do ambiente, essa aplicação deve publicar interesses na temperatura (context.ambient.temperature), som ambiente (context.ambient.sound-

intensity), luz ambiente (context.ambient.light), atividade atual do usuário (context.user.activity) e previsão do tempo(context.ambient.weather). Internamente, o LoCCAM, por sua vez, deve possuir um `CAC` para satisfazer cada interesse citado, respectivamente, CAC1, CAC2, CAC3, CAC4 e CAC5. A atividade atual do usuário diz, por exemplo, se o usuário está parado, caminhando ou andando de bicicleta. Essa informação contextual pode ser adquirida por meio de inferências realizadas sobre o acelerômetro do dispositivo. Já a previsão do tempo é adquirida por meio de um *webservice*.

Para cada um dos três possíveis locais onde o usuário poderia se encontrar, a aplicação possui uma lista de dicas que são sugeridas de acordo com as condições do ambiente e do usuário. Por exemplo, poderia ser sugerido ao usuário beber água em um dia quente, baseado na informação de temperatura capturada pelo CAC1. Além disso, CareOnTheGo poderia alertar o usuário a pegar um casaco, guarda-chuva ou óculos de sol de acordo com a previsão do tempo (CAC5). Em casa ou na rua, o usuário poderia ser alertado quando o som ambiente fosse prejudicial à sua audição (CAC2). No trabalho, CareOnTheGo poderia sugerir uma pausa para caminhar um pouco, caso o usuário estivesse sentado há muito tempo (CAC4) ou aconselhar um local com melhor iluminação para leitura (CAC3). A Figura 5.6 apresenta uma captura de tela de um protótipo não funcional (i.e. apenas interface gráfica implementada), de CareOnTheGo, apresentado como seriam exibidas as dicas contextuais.

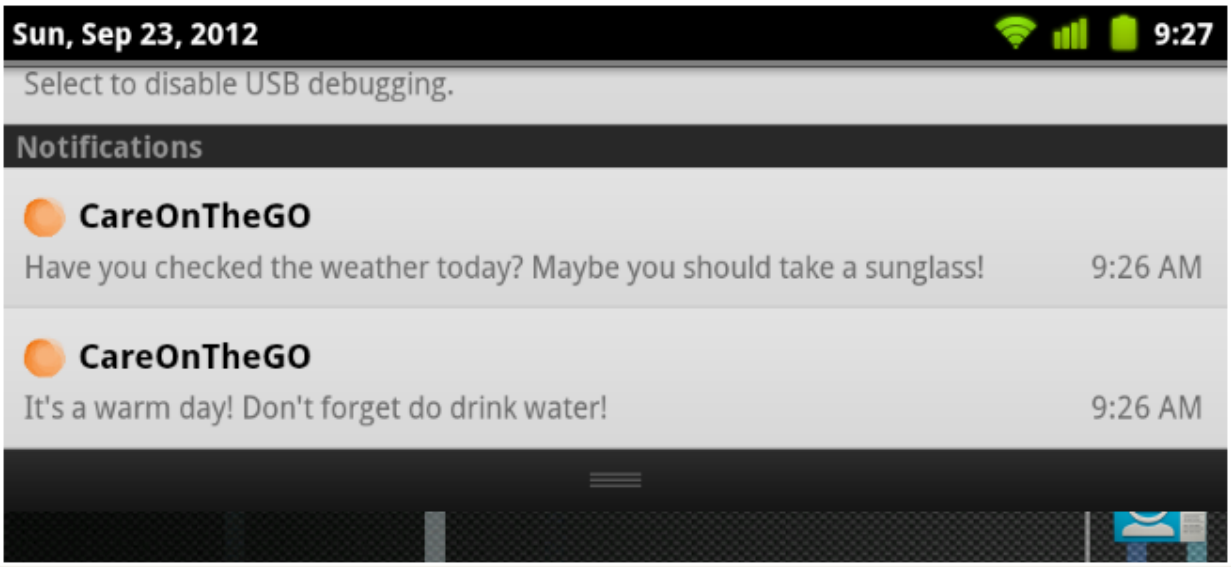

Figura 5.6: Captura de tela do protótipo não funcional do CareOnTheGo.

Durante a execução da aplicação CareOnTheGo, o interesse em informações contextuais pode mudar, disparando adaptações, de acordo com o local onde o usuário se encontra.

Os interesses da aplicação durante o perfil *home* são providos, respectivamente, pelo CAC1, CAC2 e CAC5. No perfil *work* os interesses são atendidos pelo CAC1, CAC3 e CAC4. Por fim, durante o perfil *outdoor* são utilizados os CAC1 e CAC2. A Figura 5.7 apresenta a relação entre cada perfil, seus respectivos interesses e quais componentes de aquisição de contexto os suprem.

Perfil	Interesses	CAC
home	context.ambient.temperature	CAC1
	context.ambient.sound-intensity	CAC2
	context.ambient.weather	CAC5
work	context.ambient.light	CAC1
	context.ambient.sound	CAC3
	context.user.activity	CAC4
outdoor	context.ambient.temperature	CAC1
	context.ambient.sound-intensity	CAC2

Figura 5.7: Relação entre perfis e interesses contextuais.

5.2.1 Descrição do Experimento

Na avaliação do algoritmo de adaptação, foi desenvolvido um protótipo da aplicação CareOnTheGo. Para atender os interesses contextuais do protótipo, foram desenvolvidos três `CACs` reais, CAC1, CAC3 e CAC5, e dois que simulavam a aquisição de contexto, CAC2 e CAC4. O CAC2, que deve capturar o som ambiente e publicar qual a intensidade do som, foi simulado a partir da simples captura do som ambiente. Já o CAC4, que baseado no acelerômetro informa se o usuário está há muito tempo sentado, foi simulado pela leitura do sensor de acelerômetro do dispositivo. No protótipo criado, a mudança entre os perfis *home work* e *outdoor* era realizada programaticamente.

Em cima do protótipo desenvolvido, foi conduzido um experimento para analisar o mecanismo de adaptação do *framework* proposto e o seu impacto nos recursos do dispositivo. O experimento consistia em simular um cenário onde um usuário realizava a seguinte trajetória: *home*, *outdoor*, *work*, *outdoor*, *home*. Ao longo do caminho simulado, CareOnTheGo mudava seus interesses por informações contextuais quatro vezes, disparando a adaptação de aquisição contextual do *framework*. O mesmo experimento também foi realizado desligando o mecanismo de adaptação, para comparação dos benefícios e desvantagens de cada abordagem.

Para simular o cenário descrito acima, CareOnTheGo foi executado durante 2 minutos e 30 segundos, trocando o perfil a cada 30 segundos. Ao iniciar o teste, o protótipo encontrava-se no perfil *home*. Após os primeiros 30 segundos, a primeira troca foi efetuada e assim por diante. Durante o experimento foram observados o total de memória livre no dispositivo, a porcentagem utilizada do processador e quanto tempo durava cada adaptação. O experimento foi repetido 10 vezes para cada abordagem (com e sem adaptação). Antes do início da execução do experimento, para cada abordagem, o dispositivo utilizado foi reiniciado, garantindo assim que outras aplicações do celular que estivessem executando em *background* fossem o mais semelhante possível. Apesar do experimento não refletir o comportamento real de um usuário, ele foi suficiente para reproduzir as transições e adaptações ocorridas entre os locais representados por perfis.

5.2.2 Material utilizado

O experimento descrito foi realizado em um *smartphone* Motorola Defy com Android na versão 2.3 CyanogenMod. Esse dispositivo possui um *chipset* TI OMAP 3610 e sua CPU é um Cortex-A8 de 800MHz. A memória RAM do *smartphone* é de 512MB.

5.2.3 Resultados

A Figura 5.8 apresenta um gráfico comparando o resultado médio obtido com e sem a adaptação. É possível observar um crescimento na quantidade de memória livre do sistema quando a adaptação da aquisição de contexto ocorre, já que os sensores que não estão em uso são desativados. Quanto ao uso da CPU do dispositivo, foi detectada uma pequena vantagem no uso da adaptação. O uso total da CPU durante o experimento com a adaptação foi em média 5.6%, enquanto sem adaptação foi de 6.9%. O algoritmo de adaptação levou de 20 a 1300 milissegundos para ser executado.

Os resultados obtidos demonstram que a utilização da adaptação composicional dinâmica no gerenciamento da aquisição de contexto de fato auxilia na economia de recursos do dispositivo. Além disso, com a diminuição na utilização da memória e da CPU, a bateria do dispositivo, por ser menos utilizada, também ganha uma maior autonomia. Espera-se, ainda, que em aplicações que utilizem `CACs` que demandem mais recursos, como por exemplo, um que envolva o uso do GPS, ou em casos onde várias aplicações utilizam o LoCCAM concorren-

temente, a economia de recursos alcançada seja ainda maior.

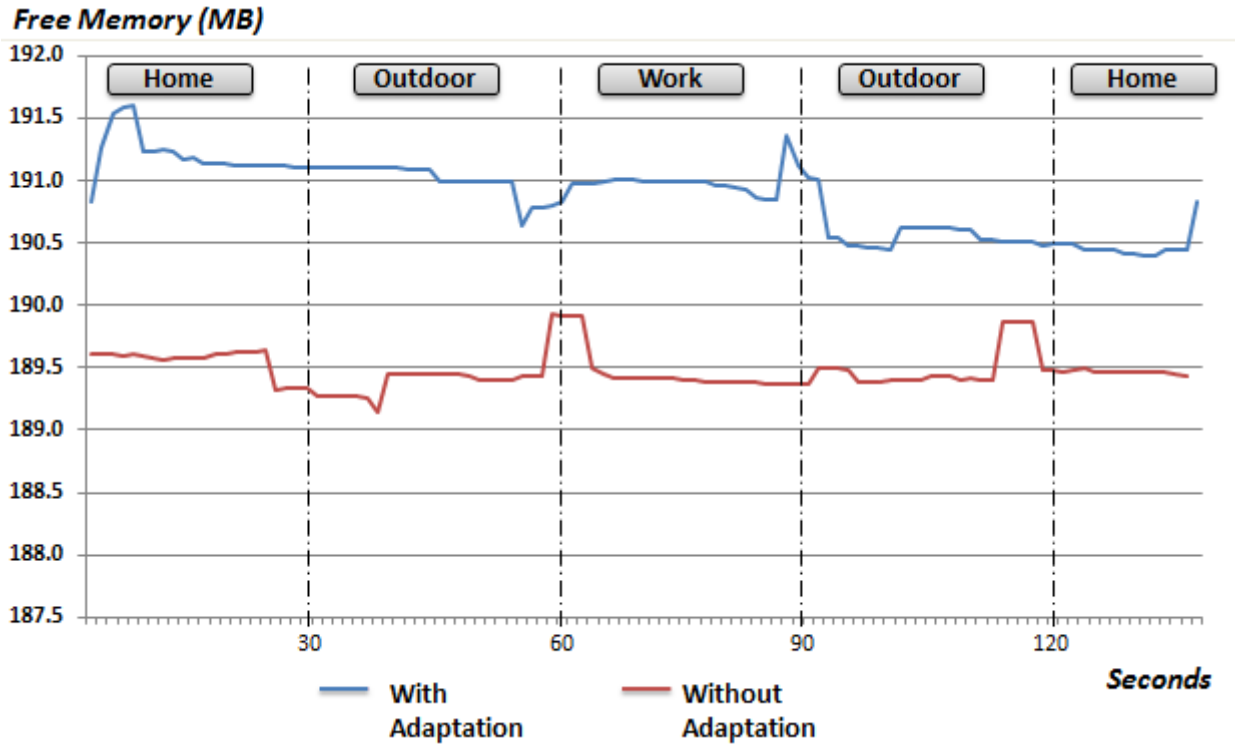

Figura 5.8: Resultado comparando execuções do LoCCAM.

5.3 Conclusão

Este capítulo apresentou uma aplicação móvel e sensível ao contexto desenvolvida sobre o LoCCAM como uma prova de conceito e testes realizados no mecanismo de adaptação dinâmica do *framework*, através de um protótipo de uma outra aplicação idealizada.

Na prova de conceito, foi demonstrado, passo a passo, como desenvolver uma aplicação móvel e sensível ao contexto utilizando o LoCCAM. Como primeiro passo, é explicado como uma aplicação publica o seus interesses contextuais no LoCCAM para que o *framework* proposto inicie a aquisição e publicação de contexto. Depois, foi apresentado como é possível subscrever a aplicação para ser notificada quando critérios, representados por *templates* e filtros, são satisfeitos. Da mesma forma, também foi demonstrado como realizar consultas síncronas de informações contextuais fornecidas pelo LoCCAM.

Com a prova de conceito, é possível perceber como o *framework* de aquisição de conceito e o LoCCAM são desacoplados do código das aplicações que os utilizam. Esse benefício é alcançado devido ao fato de na criação de uma aplicação, o desenvolvedor não escrever

códigos de acesso a sensores. Tal característica também simplifica a criação dessas aplicações, uma vez que os envolvidos na criação das mesmas não necessitam de conhecimento específicos sobre os sensores. Por fim, a prova de conceito também demonstrou como praticar o reúso de filtros previamente implementados, como no caso do PolygonFilter.

Além da prova de conceito, também foi implementado um protótipo de uma aplicação idealizada, chamada CareOnTheGo, para realização de testes no mecanismo de adaptação dinâmica. O experimento consistia em mudar o interesse contextual de uma aplicação que utilizava o LoCCAM repetidas vezes para verificar se as adaptações realizadas representariam um ganho em economia de recursos do dispositivo.

No experimento, foi detectado um ganho em economia da memória primária (memória RAM) do dispositivo, assim como uma diminuição no uso médio do processador. Também foi avaliado o tempo levado para realizar cada adaptação nos experimentos, que variou de 20 a 1300 milissegundos.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo resume o que foi discutido e desenvolvido nesta dissertação de mestrado. A Seção 6.1 apresenta os resultados alcançados com a realização da proposta, enquanto na Seção 6.2 são apresentadas quais as principais limitações encontradas. A Seção 6.3 apresenta as publicações decorrentes e a Seção 6.4 discorre sobre os trabalhos futuros que surgiram como possível continuação deste trabalho.

6.1 Resultados Alcançados

Este trabalho apresentou um *framework* de aquisição de contexto adaptável chamado CAM (Context Acquisition Manager). Esse *framework* propõe trazer a adaptação de *software*, necessária em aplicações móveis e sensíveis ao contexto, para a própria infraestrutura de gerenciamento de contexto, mais especificamente para a aquisição de contexto.

Do ponto de vista temporal, dois tipos de adaptações são permitidas pelo *framework* proposto. A primeira delas, a estática, ocorre no momento da implantação do mesmo. Através dela, é possível que o desenvolvedor escolha quais componentes do *framework*, bem como quais funcionalidades, serão instaladas no dispositivo móvel. Além de permitir uma economia de recursos, tal característica também possibilita uma maior compatibilidade entre o *framework* e dispositivos móveis. A economia de recursos se deve ao fato de que apenas os recursos necessários para execução em um dispositivo móvel estarão instalados e em execução. Já a compatibilidade se deve ao fato de que trechos do *framework* que necessitem funcionalidades não disponíveis em um dispositivo podem ser descartadas no momento da implantação. Por exemplo, uma funcionalidade que necessite do GPS pode não ser incluída, caso o dispositivo não possua esse tipo de sensor.

A outra adaptação permitida pelo *framework* é a dinâmica. Essa também permite uma economia de recursos do dispositivo, pois componentes que forneçam informações não requisitadas por aplicações que utilizam o *framework* podem ser desligados. Da mesma forma, ela também permite um bom gerenciamento de requisitos, pois novos interesses das aplicações podem ser satisfeitos em tempo de execução a partir da inicialização de componentes já instalados ou mesmo da implantação dinâmica de novos.

Nesse trabalho, foi apresentado ainda o *middleware* LoCCAM (Loosely Coupled Context Acquisition Middeware), que foi projetado por um grupo de pesquisadores do GREat. Apesar do LoCCAM não representar esse trabalho de dissertação, ele faz uso do *framework* proposto CAM para sua aquisição de contexto. Para que essa integração entre CAM e LoCCAM ocorresse, foi necessária a realização de alguns trabalhos que também são parte da contribuição dessa dissertação de mestrado, como a adaptação do SysSU e a criação de filtros espaciais.

Como prova de conceito desse trabalho de dissertação, foi apresentada a aplicação Context-Track, construída sobre o LoCCAM, e como ela foi desenvolvida. Na implementação da prova de conceito, o primeiro passo realizado foi implantar o *middleware* LoCCAM em um dispositivo. Depois foi explicado como a aplicação publicou seus interesses em informações contextuais, para que o *framework* proposto iniciasse a aquisição e publicação de contexto. O último passo da implementação, foi a subscrição realizada para que a aplicação recebesse uma notificação quando critérios definidos por ela fossem atingidos. Com a prova de conceito, foi possível apresentar como o *framework* CAM e o *middleware* LoCCAM podem ser utilizados. Além disso, também foi identificada uma clara separação entre o código de aquisição de contexto e o código das aplicações que utilizam o *framework*.

Além da prova de conceito, foi desenvolvido um protótipo de aplicação para realização de testes do mecanismo de adaptação do *framework* proposto. Com um experimento realizado, foi detectado um ganho em economia de memória e na média de uso do processador. Também foram avaliados os tempos mínimo e máximo levados para que uma adaptação ocorresse. Nos testes realizados, eles foram respectivamente de 20 milissegundos e 1300 milissegundos.

6.2 Limitações

Algumas limitações foram identificadas nesse trabalho de dissertação de mestrado. Em primeiro lugar, pelo fato da atual implementação do *framework* proposto funcionar sobre OSGi, ela só pode ser utilizada em dispositivos capazes de executar a linguagem Java. Devido a isso, plataformas de dispositivos móveis de grande aceitação no mercado, como o próprio iOS[1] da Apple, não são compatíveis.

[1] http://www.apple.com/ios/

Outra limitação se refere ao modelo de contexto utilizado. Como o *framework* CAM se propõe a ser uma infraestrutura genérica, em que qualquer aplicação sensível ao contexto poderia utilizar, seu modelo de contexto implementado também tenta ser o mais genérico possível. Isso acarreta em um modelo mais simples e que não possibilita a representação de muitos relacionamentos entre informações ou realização de inferências sofisticadas sobre ele, diminuindo o nível semântico das informações. Essa limitação pode ser contornada por aplicações pelo uso de modelos de contexto próprios sobre o modelo mais simples do *framework* proposto.

6.3 Publicações

Durante o mestrado foi possível contribuir com três publicações internacionais, todas relacionadas a proposta desta dissertação. A Tabela 6.1 apresenta as publicações. A primeira, com Qualis A1, apresenta a primeira versão do *middleware* LoCCAM, que faz uso do *framework* CAM. A segunda, de Qualis B3, também apresenta o CAM/LoCCAM, porém focando em seus filtros espaciais, uma das contribuições deste trabalho de dissertação. Por fim, a última propõe uma extensão sobre a ferramenta SysSU para que ela passe a compartilhar informações de forma distribuída. Este último trabalho permite que informações adquiridas pelo *framework* CAM em um dispositivo sejam disponibilizadas a dispositivos vizinhos ou a um servidor remoto.

Tabela 6.1: Lista de artigos publicados.

Título	Autores	Conferência	Qualis
LOCCAM - Loosely Coupled Context Acquisition Middleware	Márcio Maia, André Fonteles, Benedito Neto, Rômulo Gadelha, Windson Viana e Rossana M. C. Andrade	Symposium On Applied Computing, 2013, Coimbra, Portugal.	A1
An Adaptive Context Acquisition Framework to Support Mobile Spatial and Context-Aware Applications	André Fonteles, Benedito Neto, Márcio Maia, Windson Viana e Rossana M. C. Andrade	International Symposium on Web and Wireless Geographical Information Systems, 2013, Banff	B3
A Coordination Framework for Dynamic Adaptation in Ubiquitous Systems Based on Distributed Tuple Space	Benedito Neto, Rossana M. C. Andrade, Márcio Maia, André Fonteles e Windson Viana	9th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2013), 2013, Cagliari, Sardina, Itália.	B1

6.4 Trabalhos Futuros

Apesar do *framework* proposto já possibilitar uma economia de recursos do dispositivo, alguns pontos ainda podem ser levados em consideração no seu algoritmo de adaptação dinâmica. Além disso, um trabalho ainda deve ser desenvolvido para disponibilizar o *framework*

para a comunidade de desenvolvedores, e consequentemente receber seu suporte através da criação de CACs, filtros contextuais, etc. Entre os possíveis trabalhos futuros relacionados a essas questões, podemos citar:

- **Utilização de QoS**: Atualmente, o algoritmo de adaptação dinâmica não possui nenhum critério de desempate quando mais de um CAC satisfaz um interesse em informação contextual. Isso significa, por exemplo, que um componente com altíssimo consumo de recursos do dispositivo pode ser inicializado, ao invés de um equivalente econômico. Como trabalho futuro espera-se que o algoritmo passe a considerar a qualidade do serviço (QoS) como critério de desempate na escolha de qual componente executar.

- **Criação de Repositório Remoto**: Quando o algoritmo de adaptação não é capaz de encontrar um componente que satisfaça um interesse, sua execução para. Com a criação de um repositório de CACs remoto, é possível que o algoritmo busque uma alternativa aos componentes locais, quando esses não são capazes de suprir os interesses das aplicações. Dessa forma, valendo-se da capacidade de implantação dinâmica, o *framework* poderia fazer o *download* de um CAC a partir do repositório remoto, instalá-lo e iniciá-lo em tempo de execução para suprir um novo requisito.

- **Disponibilização de Novos CACs e Filtros**: A criação e a disponibilização para reúso de novos CACs e filtros pode diminuir o esforço de desenvolvedores de aplicações. Espera-se que sejam criados mais componentes de aquisição de contexto bem como novos filtros a serem utilizados com o SysSU.

- **Criação de um Portal**: A existência de um portal na *web* contendo o *framework* proposto, um conjunto de CACs para download e uma documentação extensa possibilitaria uma maior visibilidade e participação da comunidade de desenvolvedores no projeto. Esse portal seria de particular importância para que tal comunidade pudesse desenvolver e disponibilizar novos componentes de aquisição de contexto.

Outro trabalho futuro, menos relacionado aos temas abordados anteriormente e que demanda maior tempo para a concepção, consiste na utilização de técnicas de MDE (Model-Driven Engineering) para geração de aplicações sensíveis ao contexto que utilizem o *framework* CAM para aquisição de contexto.

Por fim, também é possível citar como trabalho futuro a utilização da versão estendida do SysSU, com suporte a computação distribuída e vista na Seção 6.3, para que vários dispositivos pudessem trocar informações contextuais adquiridas pelo *framework* CAM. Tal característica permitiria que informações que um dispositivo não fosse capaz de observar, devido a limitações de *software* e *hardware*, fossem disponibilizadas por dispositivos vizinho com capacidade de observação.

REFERÊNCIAS BIBLIOGRÁFICAS

ALLIANCE, O. *Osgi service platform, release 3*. [S.l.]: IOS Press, Inc., 2003.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, Inderscience, v. 2, n. 4, p. 263–277, 2007.

BETTINI, C.; BRDICZKA, O.; HENRICKSEN, K.; INDULSKA, J.; NICKLAS, D.; RANGANATHAN, A.; RIBONI, D. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, v. 6, n. 2, p. 161 – 180, 2010. ISSN 1574-1192. <ce:title>Context Modelling, Reasoning and Management</ce:title>. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1574119209000510>.

BRADBURY, J. S.; CORDY, J. R.; DINGEL, J.; WERMELINGER, M. A survey of self-management in dynamic software architecture specifications. In: *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. New York, NY, USA: ACM, 2004. (WOSS '04), p. 28–33. ISBN 1-58113-989-6. Disponível em: <http://doi.acm.org/10.1145/1075405.1075411>.

BRAGA, R.; de Moraes Medeiros da Costa, S.; CARVALHO, W. de; de Castro Andrade, R. M.; MARTIN, H. CAPTAIN: A context-aware web content generator based on personal tracking. In: *Web and Wireless Geographical Information Systems*. Springer, 2012. p. 134–150. Disponível em: <http://www.springerlink.com/index/0U2052W26025Q545.pdf>.

BRUNEO, D.; PULIAFITO, A.; SCARPA, M. Mobile middleware: Definition and motivations. *The Handbook of Mobile Middleware*, Auerbach Pub, p. 145–167, 2007.

CARRIERO, N.; GELERNTER, D. Linda in context. *Commun. ACM*, ACM, New York, NY, USA, v. 32, n. 4, p. 444–458, abr. 1989. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/63334.63337>.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to algorithms*. third edition. [S.l.]: MIT press, 2009.

CRNKOVIC, I.; SENTILLES, S.; VULGARAKIS, A.; CHAUDRON, M. A classification framework for software component models. *Software Engineering, IEEE Transactions on*, v. 37, n. 5, p. 593 –615, sept.-oct. 2011. ISSN 0098-5589.

DEY, A. K.; ABOWD, G. D. Towards a better understanding of context and context-awareness. In: GELLERSEN, H.-W. (Ed.). *Handheld and Ubiquitous Computing*. [S.l.]: Springer Berlin Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1707). p. 304–307. ISBN 978-3-540-66550-2.

DEY, A. K.; ABOWD, G. D.; SALBER, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction*, v. 16, n. 2-4, p. 97–166, 2001.

EUGSTER, P.; FELBER, P.; GUERRAOU, R.; KERMARREC, A.-M. The many faces of publish/subscribe. *ACM Computing Surveys*, v. 35, n. 2, p. 114–131, jun. 2003. ISSN

03600300. Disponível em: <http://portal.acm.org/citation.cfm?doid=857076.857078 http://dl.acm.org/citation.cfm?id=857078>.

FAHY, P.; CLARKE, S. Cass – a middleware for mobile context-aware applications. In: *Workshop on Context Awareness, MobiSys*. [S.l.: s.n.], 2004.

FONSECA, H. A. C. L. *Um middleware baseado em componentes para adaptação dinâmica na plataforma Android*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática, 2009.

GROUP, I. O. M. *Common Object Request Broker: Architecture and Specification*. [S.l.]: Qed Information Sciences, 1995.

GU, T.; PUNG, H. K.; ZHANG, D. Q. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, v. 28, n. 1, p. 1 – 18, 2005. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804504000451>.

HEINEMAN, G.; COUNCILL, W. *Component-based software engineering: putting the pieces together*. [S.l.]: Addison-Wesley USA, 2001.

HONG, J. yi; SUH, E. ho; KIM, S.-J. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, v. 36, n. 4, p. 8509 – 8522, 2009. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417408007574>.

INDULSKA, J.; SUTTON, P. Location management in pervasive systems. In: *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003. (ACSW Frontiers '03), p. 143–151. ISBN 1-920682-00-7. Disponível em: <http://dl.acm.org/citation.cfm?id=827987.828003>.

JOHNSON, R. E. Frameworks = (components + patterns). *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 10, p. 39–42, out. 1997. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/262793.262799>.

KAKOUSIS, K.; PASPALLIS, N.; PAPADOPOULOS, G. A. A survey of software adaptation in mobile and ubiquitous computing. *Enterprise Information Systems*, v. 4, n. 4, p. 355–389, 2010. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/17517575.2010.509814>.

KELING, D.; DALMAU, M.; ROOSE, P. A survey of adaptation systems. *International Journal*, v. 2, p. 123–140, 2012.

KETFI, A.; BELKHATIR, N.; CUNIN, P. Automatic adaptation of component-based software. In: PDPTA. [S.l.], 2002.

LAU, K.-K.; WANG, Z. Software component models. *Software Engineering, IEEE Transactions on*, v. 33, n. 10, p. 709 –724, oct. 2007. ISSN 0098-5589.

LEE, Y.; IYENGAR, S. S.; MIN, C.; JU, Y.; KANG, S.; PARK, T.; LEE, J.; RHEE, Y.; SONG, J. Mobicon: a mobile context-monitoring platform. *Commun. ACM*, ACM, New York, NY, USA, v. 55, n. 3, p. 54–65, mar. 2012. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/2093548.2093567>.

LIMA, F.; ROCHA, L.; MAIA, P.; ANDRADE, R. M. C. A decoupled and interoperable architecture for coordination in ubiquitous systems. In: IEEE. *Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on*. [S.l.], 2011. p. 31–40.

MAIA, M. E.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. Loccam-loosely coupled context acquisition middleware. In: ACM. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. [S.l.], 2013. p. 534–541.

MARINHO, F. G.; ANDRADE, R. M. C.; WERNER, C.; VIANA, W.; MAIA, M. E.; ROCHA, L. S.; TEIXEIRA, E.; FILHO, J. B. F.; DANTAS, V. L.; LIMA, F.; AGUIAR, S. Mobiline: A nested software product line for the domain of mobile and context-aware applications. *Science of Computer Programming*, n. 0, p. –, 2012. ISSN 0167-6423. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167642312000871>.

MCKINLEY, P.; SADJADI, S.; KASTEN, E.; CHENG, B. Composing adaptive software. *Computer*, v. 37, n. 7, p. 56 – 64, july 2004. ISSN 0018-9162.

MITCHELL, M.; MEYERS, C.; WANG, A.-I.; TYSON, G. Contextprovider: Context awareness for medical monitoring applications. In: *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*. [S.l.: s.n.], 2011. p. 5244 –5247. ISSN 1557-170X.

PREUVENEERS, D.; BERBERS, Y. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In: *Symposium on Applied Computing: Proceedings of the 2007 ACM symposium on Applied computing*. [S.l.: s.n.], 2007. v. 11, n. 15, p. 1165–1170.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, maio 2011. ISSN 1556-4665. Disponível em: <http://doi.acm.org/10.1145/1516533.1516538>.

STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: *First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp*. [S.l.: s.n.], 2004.

SZYPERSKI, C.; GRUNTZ, D.; MURER, S. *Component software: beyond object-oriented programming*. [S.l.]: Addison-Wesley, 2002.

VIANA, W.; FILHO, J.; GENSEL, J.; OLIVER, M. V.; MARTIN, H. Photomap–automatic spatiotemporal annotation for mobile photos. *Web and Wireless Geographical Information Systems*, Springer, p. 187–201, 2007.

VIANA, W. C. *Mobilité et sensibilité au contexte pour la gestion de documments multimédias personnels: CoMMediA*. Tese (Doutorado) — Université Joseph-Fourier - Grenoble, 2010. Disponível em: <http://hal.archives-ouvertes.fr/tel-00499550/>.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Designing context-sensitive systems: An integrated approach. *Expert Systems with Applications*, v. 38, n. 2, p. 1119–1138, 2011. ISSN 0957-4174. <ce:title>Intelligent Collaboration and Design</ce:title>. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417410004173>.

WANG, A.; QIAN, K. *Component-oriented programming*. [S.l.]: Wiley-Interscience, 2005.

WEISER, M. The computer for the 21st century. *Scientific American*, New York, v. 265, n. 3, p. 94–104, 1991.

WILLIAMS, S.; KINDEL, C. *The component object model: A technical overview*. [S.l.], 1994.

APÊNDICE A – DIAGRAMA DE CLASSES SIMPLIFICADO DO LOCCAM SEM CACS