
**Universidade Federal do Ceará**
**Centro de Ciências**
**Departamento de Computação**
**Mestrado e Doutorado em Ciência da Computação**

**Dissertação de Mestrado**

# Uma Abordagem Dirigida por Modelos para a Configuração de Aquisição de Contexto Intermediada por *Middleware*

**Paulo Artur de Sousa Duarte**

Fortaleza – Ceará
2014

**Paulo Artur de Sousa Duarte**

# Uma Abordagem Dirigida por Modelos para a Configuração de Aquisição de Contexto Intermediada por *Middleware*

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Orientador: Windson Viana de Carvalho, Dr.

Co-Orientador: Fernando Antonio Mota Trinta, Dr.

Fortaleza – Ceará
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

D874a Duarte, Paulo Artur de Sousa.
Uma abordagem dirigida por modelos para a configuração de aquisição de contexto intermediada por middleware / Paulo Artur de Sousa Duarte. – 2014.
138 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2014.
Orientação: Prof. Dr. Windson Viana de Carvalho.
Coorientação: Prof. Dr. Fernando Antonio Mota Trinta.

1. Sensibilidade ao contexto . 2. MDE . 3. Middleware. 4. DSL. I. Título.

CDD 005

---

# Uma Abordagem Dirigida por Modelos para a Configuração de Aquisição de Contexto Intermediada por *Middleware*

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em ___/___/_____

Banca Examinadora

_____

Prof. Windson Viana de Carvalho, Dsc. (Orientador)
Universidade Federal do Ceará – UFC

_____

Prof. Fernando Antonio Mota Trinta, Dsc. (Co-Orientador)
Universidade Federal do Ceará – UFC

_____

Prof. Cidcley Teixeira de Souza, Dsc.
Instituto Federal de Educação, Ciência e Tecnologia do Ceará – IFCE

_____

Prof. Vinicius Cardoso Garcia, Dsc.
Universidade Federal de Pernambuco – UFPE

_____

Prof. Lincoln Souza Rocha, Dsc.
Universidade Federal do Ceará – UFC

Dedico essa dissertação aos meus pais, José Artur e Heliete.

# Agradecimentos

Aos meus pais, José Artur e Heliete, e à minha tia, Mariana, peças fundamentais na minha vida, e que sempre me apoiaram em tudo.

Ao meu orientador, professor Windson Viana, por todos os conselhos, ajuda, paciência, palavras de incentivo, brincadeiras e puxões de orelha. Por todas as dúvidas tiradas, por todas as sugestões e por todo o apoio durante todo esse tempo.

Ao meu co-orientador, professor Fernando Trinta, pelas sugestões, conselhos, opiniões, ajuda, pelas dúvidas tiradas e pela paciência.

À banca da dissertação. Ao professor Cidcley Teixeira, meu primeiro orientador, por mais de três anos no IFCE, e responsável por me fazer querer trilhar o caminho da pesquisa. Ao professor Lincoln Rocha, por todos os conselhos dados no decorrer do mestrado. E ao o professor Vinícius Garcia, por estar presente na banca da minha defesa.

À professora Rossana Andrade, por ter me recebido tão bem no GREat. E ao pessoal do GREat, Darilu, Géssika, Janaína, Ronaldo, e o pessoal do Suporte, por toda a ajuda durante esses anos.

Ao Felipe Mota, bolsista de Iniciação Científica, por todo o auxílio e conselhos no entendimento do LoCCAM e na lógica das regras de transformação. Ao Anderson Almada, bolsista de Iniciação Científica, por toda a ajuda no desenvolvimento dos Instaladores da CRITiCAL. Além de um agradecimento especial a ambos pelas melhorias realizadas no LoCCAM durante o último um ano e meio.

À Vanessa Viana, por todas as vanessices feitas em todos os anos de amizade e mestrado e a coautoria no padrão apresentado no MiniPlop.

Ao casal de amigos, Gabriela Bruno e Bruno Ribeiro, por toda a amizade, apoio e hospitalidade que me deram.

À Natacha Queiroz, por passar o ano de 2014 inteiro perguntando quase semanalmente quando a minha defesa seria marcada.

Aos amigos da pós-graduação, em especial Rodrigo “Terebet” Teles, Rainara Maia, Ítalo Linhares, Larissa Castro, Thalisson Oliveira, Nayane Ponte, Ricardo Viana, Andressa Bezerra, Ismayle Santos, Deborah Magalhães e Christiano Rodrigues. Pois boa parte de nossa

força e perseverança nos momentos difíceis na pós-graduação reside no fato de sabermos que nunca estivemos sozinhos.

E a todos meus outros amigos que, de alguma forma, contribuíram para a realização deste trabalho. Em especial, mencionar Ana Paula, Dayanny, Nágela, Elieyde, Gisele, Edigleison, Natália, Thaís, Alice, Airton, entre outros.

Ao meu gerente, Davi Albuquerque, pela flexibilidade dos meus horários, pelos conselhos e pela ajuda, como um desenvolvedor com experiência com o LoCCAM, em sugerir melhorias tanto na ferramenta quanto no LoCCAM.

Ao Marcio Maia, pela ajuda na definição do meta-modelo e em alguns entraves que tive na lógica de algumas regras de transformação.

Ao Reinaldo Braga e Emanuel Coutinho, pelos conselhos que resultaram na avaliação de desempenho.

À Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP), pela bolsa concedida, possibilitando a dedicação exclusiva para a execução do projeto de pesquisa.

“Um Jedi deve um compromisso total assumir.
A mente mais séria. (...) Aventura? Diversão?
Um Jedi por essas coisas não anseia.”

Mestre Yoda, falando de

Jedis e Pós-Graduandos

# Resumo

As aplicações móveis e sensíveis ao contexto se caracterizam por capturar a informação contextual do ambiente em que estão inseridas para utilizá-la de acordo com um objetivo específico (e.g., adaptação de interface, recomendação de conteúdo). A heterogeneidade dos dispositivos, a limitação de recursos dos mesmos e o acesso a sensores aumentam a complexidade do desenvolvimento deste tipo de software. Uma forma recente de se lidar com tais problemas de desenvolvimento é a adoção combinada de plataformas de *middleware* com os princípios do paradigma MDE (*Model-Driven Engineering*). O objetivo é tanto de reduzir o tempo de desenvolvimento com a geração de código a partir de modelos como de aumentar as potencialidades dos aplicativos através do uso dos serviços providos pelas plataformas de middleware. Seguindo essa tendência, essa dissertação propõe uma abordagem de geração de código de configuração de aquisição de contexto em aplicações móveis Android com duas principais contribuições: a DSL (*Domain-Specific Language*) ContextRuleML e a Ferramenta CRITiCAL. A DSL ContextRuleML permite ao um desenvolvedor modelar em alto nível as informações contextuais e as regras contextuais de uma aplicação sensível ao contexto. A CRITiCAL é uma ferramenta de configuração para aplicações móveis que utiliza os modelos produzidos com a DSL para a geração de um esqueleto inicial de um aplicativo móvel e sensível ao contexto. O código gerado invoca métodos de uma plataforma de *middleware* de aquisição de contexto: o LoCCAM. A ferramenta também gera a configuração e a instalação automatizada do LoCCAM e de seus componentes com base no modelo gerado. A validação da abordagem compreendeu duas etapas. A primeira foi uma avaliação de usabilidade da ferramenta realizada com 14 voluntários da área da computação. A segunda foi uma avaliação de desempenho que comparava em termos de tempo de resposta e memória utilizada o código gerado pela CRITiCAL com o código escrito de forma manual por um especialista em desenvolvimento utilizando o LoCCAM.

**Palavras-chave:** Sensibilidade ao contexto, MDE, *Middleware*, DSL

# Abstract

Context-aware mobile (CAM) applications retrieve contextual information from the environment in which they run in order to achieve a specific goal (e.g., interface adaptation, content recommendation). Features such as device heterogeneity, scarce resources and sensors diversity improve the complexity of development to this kind of software. Recent way to deal with such development issues is the adoption of two combined approaches: (i) middleware platforms and (ii) principles of MDE paradigm (e.g., Model-Driven Engineering). These approaches aim at reducing the total development time of CAM applications by using code generation from higher-level models, and increasing the potential applications through the use of services provide by middleware platforms. Thus, this dissertation proposes an approach to generate code that configures the acquisition of context in Android mobile applications. Based on MDE, we created the DSL (Domain-Specific Language) ContextRuleML, which allows the developer to model the contextual rules of a CAM application in a higher-level notation. A configuration tool, named CRiTICAL, uses the models produced with the DSL to generate an initial structure of a CAM application that uses an existing middleware platform to acquisition of context called LoCCAM. The tool also generates the configuration and the automated installation of LoCCAM and its components based on generated model. Two evaluations were made. First, a usability evaluation was realized by 14 volunteers. Second, a performance evaluation compares response time and memory used by the code generated by CRITiCAL against code written by a specialist in development with LoCCAM, without using our solution.

**Keywords:** Context-Awareness, MDE, Middleware, DSL

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas

| | |
|---|---|
| AMUSE | *Agent-based Middleware for Ubiquitous Service Environment* |
| ATL | *ATL Transformation Language* |
| BPEL | *Business Process Execution Language* |
| CAC | Componente de Aquisição de Contexto |
| CAM | *Context Acquisition Manager* |
| CIM | *Computation Independent Model* |
| CK | *Context Key* |
| COPAL | *COntext Provisioning for ALl* |
| CRITiCAL | *ConfiguRatIon Tool for Context Aware and mobiLe applications* |
| DSL | *Domain Specific Language* |
| ECA-DL | *Events-Conditions-Actions Domain Language* |
| EMF | *Eclipse Modeling Framework* |
| GEF | *Graphical Editing Framework* |
| GMF | *Graphical Modeling Framework* |
| GReAT | *Graph Rewriting and Transformation* |
| GREat | Grupo de Redes de Computadores, Engenharia de Software e Sistemas |
| IDE | *Integrated Development Environment* |
| IDSL | *Interaction System Design Language* |
| LoCCAM | *Loosely Coupled Context Acquisition Middleware* |
| M2M | *Model to Model* |
| M2T | *Model to Text* |
| MDA | *Model-Driven Architecture* |
| MDD | *Model-Driven Development* |
| MDE | *Model-Driven Engineering* |
| MDSD | *Model-Driven Software Development* |
| MOF | *Meta-Object Facility* |
| OMG | *Object Management Group* |
| PIM | *Platform Independent Model* |
| PSM | *Platform Specific Model* |
| SysSU | *System Support for Ubiquity* |
| QVT | *Query/View/Transformation* |
| UML | *Unified Modeling Language* |
| XMI | *XML Metadata Interchange* |

# Capítulo 1

# Introdução

Esta dissertação apresenta uma abordagem para auxiliar o desenvolvimento de aplicações móveis e sensíveis ao contexto. Ela se baseia na modelagem e posterior geração do código que encapsula o comportamento das aquisições de contexto de uma aplicação. A parte central da abordagem é a ferramenta CRITiCAL (*ConfiguRation Tool for Context Aware and mobiLe applications*), que permite a modelagem das informações e regras contextuais e geração posterior do código correspondente ao modelo construído.

Este capítulo está estruturado da seguinte forma: a Seção 1.1 realiza a contextualização e a caracterização do problema que esta dissertação aborda. Na Seção 1.2, é apresentada a motivação que levou ao desenvolvimento da abordagem apresentada neste trabalho. Na Seção 1.3 são expostos os objetivos e as contribuições principais deste trabalho. Na Seção 1.4, é descrita a metodologia utilizada neste trabalho. Finalizando, na Seção 1.5 é descrita a organização do restante desta dissertação.

## 1.1 Contextualização e Caracterização do Problema

Os últimos anos apresentaram uma grande popularização do uso de dispositivos móveis (*tablets*, *celulares*, *smartphones*, entre outros). Esse processo aconteceu concomitantemente, ou por consequência direta, com os diversos avanços na pesquisa das áreas de computação móvel e redes sem fio. A evolução e a popularização dos dispositivos móveis acabaram gerando maior demanda no desenvolvimento de aplicações voltadas para esses dispositivos. O aumento do poder de processamento também possibilitou a evolução e a popularização de uma categoria de aplicações, as que unem os conceitos de mobilidade e sensibilidade ao contexto. Estas podem ser descritas como sendo aplicações que conseguem receber informações do ambiente no qual o dispositivo está inserido e reagir ou se adaptar às mudanças que possam vir a ocorrer no ambiente (e.g. adaptação do grau de brilho da tela de acordo com luminosidade do ambiente em que o dispositivo está inserido).

As aplicações móveis e sensíveis ao contexto fazem parte do cenário idealizado por Mark Weiser no artigo "*The Computer for the 21st Century*" há duas décadas, no qual foi

introduzido o termo Computação Ubíqua [WEISER, 1991]. Muitas das características apresentadas por Weiser já estão perceptíveis, em maior ou menor grau, nos *tablets* e *smartphones* atuais.

Todavia, o desenvolvimento de software dessa natureza ainda é repleto de problemas e desafios. A grande variedade de dispositivos, plataformas de desenvolvimento, protocolos e interfaces de comunicação acabam dificultando o desenvolvimento desse tipo de aplicações. Essas plataformas e dispositivos são muitas vezes incompatíveis em termos de suporte de software. Um exemplo de problema causado por essa heterogeneidade é o problema da fragmentação do software, ocorrida mais notadamente na plataforma Android. Essa plataforma se caracteriza por possuir diversas versões do sistema operacional que ainda estão implantadas em dispositivos comerciais. E, muitas vezes, aplicações desenvolvidas para uma versão são incompatíveis para outras. Isso traz acréscimos nas despesas inerentes ao desenvolvimento de tais aplicações devido à necessidade de criar versões diferentes, compatíveis com as especificações de cada dispositivo ou API. A Figura 1.1 ilustra o grau desse problema, mostrando a diversidade de dispositivos móveis com o sistema operacional Android, proporcionalmente ao número de unidades de cada modelo.

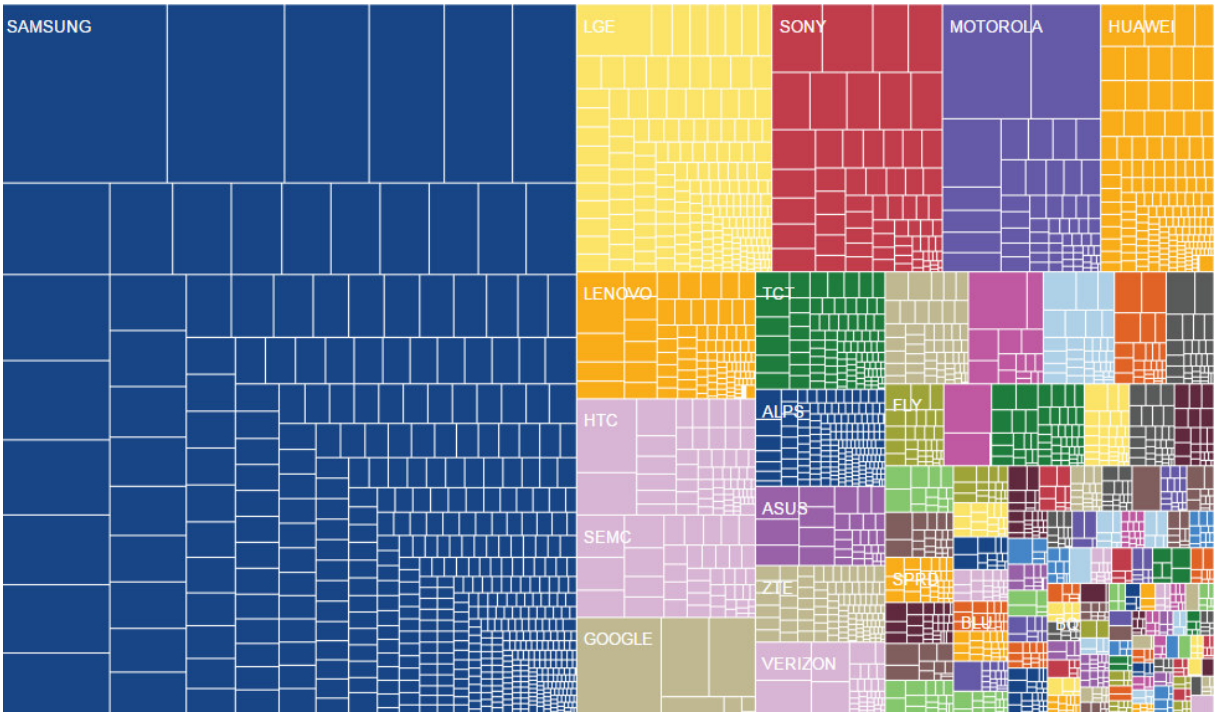


**Figura 1.1 – Fragmentação do Android pelos dispositivos de cada empresa em agosto de 2014 [OPEN SIGNAL, 2014]**

A Figura 1.2, por sua vez, apresenta a proporção das versões do Android que ainda estão em uso, de acordo com os dados obtidos pela [OPEN SIGNAL, 2014]. Com o crescente

avanço na tecnologia, muitas vezes aplicações criadas para as APIs mais novas acabam se tornando incompatíveis em versões mais antigas. Bem como o contrário também pode acontecer e aplicações desenvolvidas em versões mais antigas requererem profundas modificações para poderem ser executadas nas versões mais recentes.

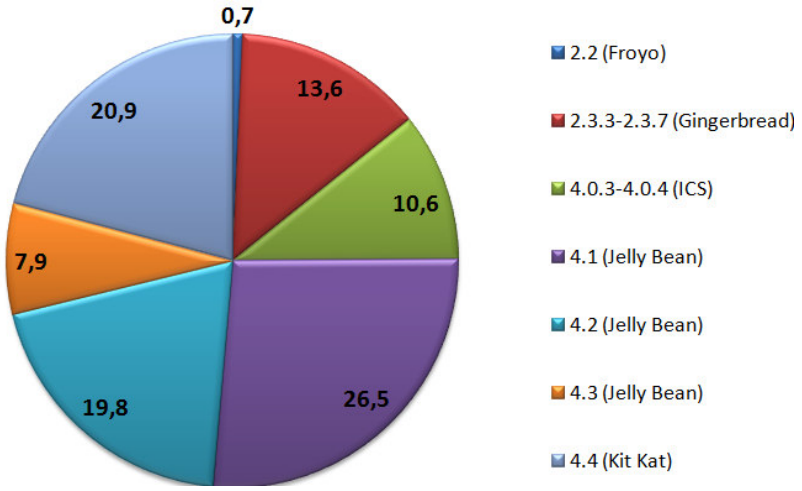


**Figura 1.2 – Proporção existente do uso das versões do SO Android em agosto de 2014 [adaptado de OPEN SIGNAL, 2014]**

Por tudo o que foi apresentado, percebe-se que a necessidade de uma maior portabilidade e interoperabilidade dos aplicativos para a redução dos esforços e gastos exigidos para o desenvolvimento destes. Isso sem mencionar as limitações inerentes à natureza dos dispositivos móveis, como as limitações do consumo de recursos (e.g., a energia da bateria, a memória) e a ausência de perenidade na conexão do dispositivo com a Internet.

Felizmente, já existem diversas abordagens que conseguem mitigar alguns dos problemas apresentados e prover portabilidade e interoperabilidade para os aplicativos. Uma dessas abordagens é a adoção de plataformas de *middleware*. No entanto, ela apresenta vantagens e limitações que serão apresentadas na Subseção a seguir.

## 1.2 Motivação

A adoção de plataformas de *middleware* é uma abordagem utilizada para a redução da complexidade do desenvolvimento de aplicações sensíveis ao contexto. Um *middleware* age como uma camada intermediária entre as aplicações e o sistema operacional. O seu objetivo é uniformizar a sintaxe das interfaces entre estes, simplificando o desenvolvimento [GRIGORAS, 2006]. Um exemplo de *middleware* voltado para o desenvolvimento de aplicações móveis e sensíveis ao contexto é o LoCCAM (*Loosely Coupled Context Acquisition Middleware*) [FONTELES, 2013].

O LoCCAM é um *middleware* de aquisição de contexto voltado para dispositivos móveis que utilizam o Android como sistema operacional. Dentre as suas principais características, pode-se mencionar o fraco acoplamento às aplicações que o utilizam e a possibilidade de autoadaptação dos componentes de aquisição e inferência das informações contextuais. Dentre os seus objetivos, pode-se mencionar a redução da complexidade do desenvolvimento de aplicações móveis e sensíveis ao contexto, como, por exemplo, o acesso aos dados dos sensores.

Entretanto, o desenvolvimento de aplicações com o LoCCAM apresenta alguns entraves. Dentre os quais, podem-se mencionar as dificuldades no tocante à configuração do *middleware* para um determinado dispositivo, com os componentes de aquisição de contexto tendo que ser inseridos no dispositivo de maneira *ad-hoc* pelo desenvolvedor. Além disso, a sintaxe da comunicação entre a aplicação e o LoCCAM, que prescinde que o desenvolvedor conheça certos detalhes da arquitetura do *middleware*. E também as sintaxes de certas ações utilizando o LoCCAM, tais como regras de eventos ou com composição de condições do tipo "uma OU outra", permanecem com uma complexidade de código alta.

Esses inconvenientes da programação do LoCCAM também se repetem em outras plataformas de *middleware* para aplicações sensíveis ao contexto [HENRICKSEN *et al*, 2005] [MAIA, ROCHA E ANDRADE, 2009]. Todavia, têm ganhado enfoque propostas e abordagens que mesclam a adoção de plataformas de *middleware* com conceitos de paradigmas de programação generativa como a MDE (*Model-Driven Engineering*). Essa integração segue o princípio comum e norteador das duas abordagens que é a busca pela simplificação do desenvolvimento das aplicações. Dentre essas pesquisas, destaca-se a criação de linguagens de domínio específico (DSL) dedicadas ao domínio da sensibilidade ao contexto e que são voltadas para a utilização em conjunto com plataformas de *middleware*. Enquanto as plataformas de *middleware* fornecem (ou substituem) as camadas de aquisição, distribuição e gerenciamento de contexto, as DSLs fornecem uma linguagem que provê elementos de sintaxe em alto nível, reduzem a complexidade do uso de informações contextuais e da codificação do uso das plataformas de *middleware* [ALMEIDA *et al*, 2005][Daniele *et al* [2009][LI, SEHIC e DUSTDAR, 2010][HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]

Portanto, a principal motivação para este trabalho é auxiliar o desenvolvimento de aplicações móveis e sensíveis ao contexto. Com essa finalidade, se buscou uma abordagem que una os conceitos de MDE e plataformas de *middleware*. Esta abordagem deve prover uma

DSL que permita a modelagem de informações e regras contextuais, de forma a auxiliar o desenvolvimento de aplicações compatíveis com o *middleware* LoCCAM, que será o responsável pela aquisição e gerenciamento das informações contextuais.

## 1.3 Objetivos e Contribuições

Esse trabalho de mestrado tem como objetivo propor uma abordagem para a modelagem de informações contextuais e regras contextuais com posterior geração de código com o fim de auxiliar o desenvolvimento de aplicações móveis e sensíveis ao contexto. Este objetivo principal pode ser dividido nos seguintes objetivos específicos:

- Construir uma DSL que provenha elementos de sintaxe em alto nível para a modelagem de informações contextuais de forma a oferecer a redução da complexidade do código inerente ao uso dessas informações.
- Implementar um processo de geração de código baseado nos modelos construídos pela DSL. A geração de código permite que os modelos de alto-nível criados pela DSL sejam compatíveis com a plataforma de *middleware* encarregada pela aquisição e gerenciamento do contexto.
- Desenvolver uma ferramenta que permita a modelagem das informações contextuais através da DSL e a transformação desta em código-fonte. Através dessa ferramenta, seriam unificados os processos de modelagem e geração de código.

Para atingir esses objetivos, foram propostas as seguintes etapas:

- Desenvolver um metamodelo que permita a construção de modelos representativos de informações contextuais e regras contextuais e um ambiente de modelagem visual que permita a criação destes modelos.
- Desenvolver *templates* com as regras de transformação que permitam, com base no modelo criado, a geração de um projeto Android com todos os métodos de comunicação entre a aplicação e o *middleware* LoCCAM instanciados ou a criação de uma lista de componentes de aquisição de contexto necessários pela aplicação.
- Desenvolver um mecanismo de instalação automática dos componentes de aquisição de contexto em um dispositivo móvel.
- Por fim, integrar todos os elementos desenvolvidos nas etapas anteriores em uma ferramenta intitulada CRITiCAL (*ConfiguRation Tool for Context Aware and mobiLe applications*).

## 1.4 Metodologia

A metodologia científica utilizada neste trabalho apresenta as seguintes etapas:

- **Revisão da Literatura:** Foi realizada uma revisão bibliográfica envolvendo os conceitos de Sensibilidade ao Contexto, Plataformas de *Middleware* e Engenharia Dirigida por Modelos. Também foi realizada uma revisão da literatura de trabalhos que envolviam conjuntamente a sensibilidade ao contexto com os paradigmas da engenharia dirigida por modelos.
- **Estudo dos Trabalhos Relacionados:** Por fim, o escopo foi reduzido para se focar na busca por trabalhos que envolvessem DSLs voltadas para desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizassem plataformas de *middleware*. Foi realizada uma revisão da literatura existente neste sub-tema e os trabalhos encontrados foram estudados.
- **Definição das Funcionalidades da Ferramenta:** Com base nas características encontradas nos trabalhos relacionados, foi definido um conjunto de características que englobam as características desejáveis para a ferramenta.
- **Definição do Metamodelo:** Esta etapa compreendeu a definição do metamodelo utilizado pela ferramenta proposta. Ele é baseado nas principais características encontradas nos trabalhos relacionados e é norteado pelas funcionalidades elencadas na etapa anterior.
- **Construção da Ferramenta e das Regras de Transformação:** Uma vez que o metamodelo foi criado, pode-se partir para a criação da ferramenta CRITiCAL. Concomitantemente, foram definidas as regras de transformação que são responsáveis pela geração do código a partir do modelo. Ao final desta etapa, as diversas funcionalidades da ferramenta foram integradas.
- **Avaliação:** Com a ferramenta concluída, pode-se realizar a avaliação para verificar se ela atende os objetivos propostos inicialmente.

## 1.5 Organização da Dissertação

Esta dissertação está organizada em seis capítulos. Sendo este o primeiro, no qual foi introduzida a contextualização do assunto abordado, a motivação e o objetivo da dissertação. Os capítulos seguintes são:

**Capítulo 2 – Fundamentação Teórica:** Tem como objetivo abordar os conceitos teóricos que compõem esse trabalho. Dentre eles, pode-se citar: a Sensibilidade ao Contexto, a Engenharia Dirigida por Modelos, as Plataformas de *Middleware* e o *middleware* LoCCAM.

**Capítulo 3 – Trabalhos Relacionados:** Apresenta abordagens existentes na Literatura acerca do tema de DSLs voltadas para plataformas de *middleware* sensíveis ao contexto.

**Capítulo 4 – CRITiCAL:** Apresenta a proposta central da dissertação, que engloba a ferramenta de modelagem CRITiCAL, a DSL ContextRuleML e a evolução realizada no *middleware* LoCCAM no decorrer do desenvolvimento da CRITiCAL.

**Capítulo 5 – Avaliação:** Apresenta as avaliações realizadas com a ferramenta CRITiCAL. São detalhados as metodologias, procedimentos e resultados obtidos tanto da avaliação de usabilidade quanto da avaliação de desempenho.

**Capítulo 6 – Conclusão:** Descreve os resultados alcançados por este trabalho, assim como as suas conclusões. Além disso, são mencionados possíveis caminhos e melhorias que podem ser realizados em trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo, são apresentados os principais conceitos teóricos relacionados a este trabalho. A Seção 2.1 apresenta os conceitos relacionados à Sensibilidade ao Contexto, abrangendo também questões como a definição de contexto adotada nesta pesquisa, a modelagem de contexto e o desenvolvimento de aplicações sensíveis ao contexto. A Seção 2.2 apresenta a abordagem das plataformas de *middleware*, focando especialmente nas plataformas de *middleware* voltadas para o desenvolvimento de aplicações móveis e sensíveis ao contexto, bem como expondo as limitações e os desafios dessa abordagem. A Seção 2.3 apresenta mais especificamente o *middleware* LoCCAM, detalhando sua arquitetura, componentes e estratégia de adaptação de aquisição contextual. A Seção 2.4 apresenta os conceitos da Engenharia Dirigida por Modelos, explicando seus princípios, vantagens, desvantagens, abordagens principais e conceitos fundamentais, tais como as Linguagens de Domínio Específico e as Transformações de Modelos.

## 2.1 Sensibilidade ao Contexto

O termo "*context-aware*" foi apresentado pela primeira vez por [SCHILIT e THEIMER, 1994]. No trabalho apresentado por estes autores, a sensibilidade ao contexto é definida como sendo "a habilidade de uma aplicação móvel em descobrir e reagir a mudanças no ambiente em que ele está situado".

Esta noção e os conceitos acerca da sensibilidade ao contexto foram aperfeiçoados com o passar dos anos. Em 2001, Dey definiu como sendo um sistema sensível ao contexto "aquele que utiliza contexto para prover informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário" [DEY, 2001]. Além dessa definição, Dey também conceitualizou o que seria "contexto" em si. Para ele, contexto é entendido como sendo "qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação em si". Essas definições passaram a ser bastante referenciadas pela academia nos anos subsequentes

[DEY, 2001].

A sensibilidade ao contexto pode ser considerada um dos requisitos básicos de um sistema ubíquo [LIMA, 2011]. De fato, aplicações que apresentam sensibilidade ao contexto fazem parte do cenário da Computação Ubíqua imaginado por Mark Weiser [WEISER, 1991]. No cenário visionário de Mark Weiser, os dispositivos computacionais são incorporados em elementos comuns utilizados pelos seres humanos. Desta forma, os dispositivos são capazes de trocar dados uns com os outros e estão tão integrados com o ambiente no qual se encontram que a interação com o usuário se torna natural e quase imperceptível [WEISER, 1991].

Através da sensibilidade ao contexto, os sistemas computacionais podem, portanto,adaptar o seu comportamento de acordo com as informações que recebem acerca da situação do ambiente ou do usuário de forma a reduzir intervenções dos usuários para adequação do seu comportamento, o que torna a interação mais natural e calma. Aplicações móveis e sensíveis ao contexto são um subconjunto dos sistemas ubíquos que são executadas em um dispositivo móvel e utilizam a sensibilidade ao contexto para se adaptar ou para aperfeiçoar suas funcionalidades.

Alguns exemplos de aplicações móveis que requerem a sensibilidade ao contexto para o seu pleno funcionamento são:

- Sistemas baseados em localização [FONTELES *et al*, 2013], aplicações que inferem a localização do dispositivo móvel no qual estão executando e, com base nesta, adaptam o serviço que oferecem;
- Sistemas de anotação de mídias [VIANA *et al*, 2014], serviços que permitem a inserção automática de metadados contextuais (e.g., localização, informação temporal) em determinadas mídias (e.g., fotos, áudios) de forma a utilizá-los para a organização posterior destes documentos;
- Sistemas de monitoramento de atividade esportiva [BRAJDIC e HARLE, 2013], aplicações que, com base em informações como localização, velocidade e outras, auxiliam os usuários em práticas esportivas.

As definições de contexto e de sensibilidade ao contexto vêm sendo modificadas ao longo dos últimos anos para incorporar novos requisitos destes sistemas. O conceito de Dey, por exemplo, foi expandido por Viana de forma a retirar a limitação da definição que confinava o contexto aos elementos que caracterizavam a interação entre usuário e sistema

[VIANA, 2010]. Por conseguinte, na definição de Viana é dito que "faz parte do contexto toda informação que pode descrever a situação das entidades (e suas relações) envolvidas em uma ação que seja julgada importante para o sistema. Estas entidades são os conceitos abstratos e os objetos físicos presentes na zona de observação do sistema em um instante tn de observação" [VIANA, 2010]. A ideia é incluir sistemas que fazem a aquisição de informações contextuais que não são relevantes no instante da captura, mas podem ser utilizadas depois para melhoria de funcionalidades do sistema (e.g., um sistema móvel de anotação de fotos, um sistema de recomendação móvel baseado em comportamento de uso)

Esta definição apresenta os conceitos de Zona de Interesse e Zona de Observação. Por Zona de Interesse, compreende-se o conjunto de entidades que o sistema julga importante para a situação daquele instante da especificação. Por zona de observação, entende-se o conjunto de entidades, bem como suas relações, que possam ser percebidas pelo sistema em um determinado instante de observação [VIANA, 2010].

Desta maneira, a definição de contexto pode ser visualmente exemplificada como sendo o conjunto de entidades (e suas relações) que formam a interseção entre a zona de interesse e a zona de observação [VIANA, 2010], conforme mostrado na Figura 2.1.

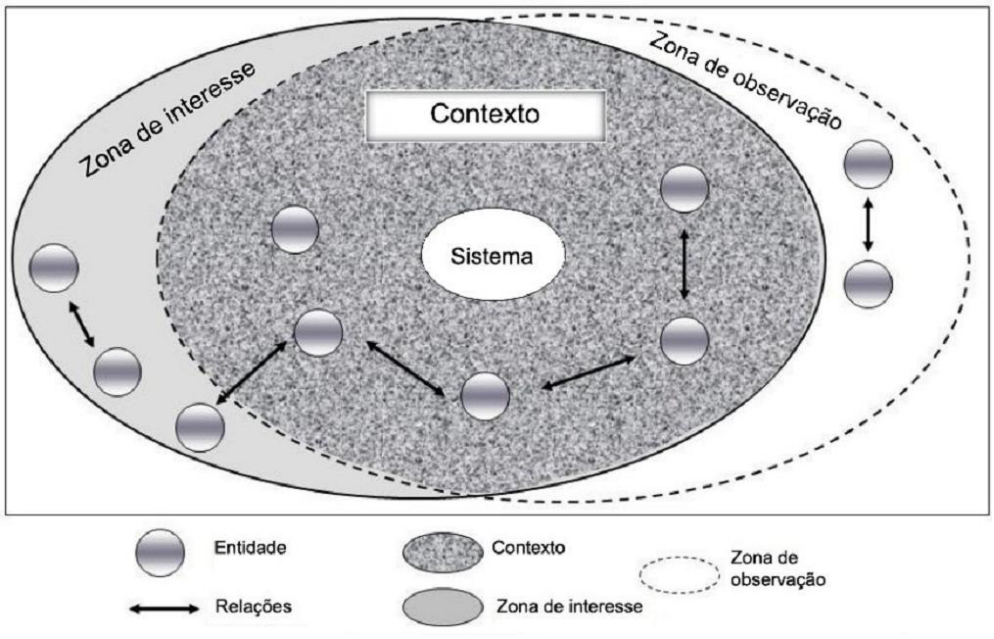


**Figura 2.1 – Definição de Contexto segundo a ZO e ZI [Adaptado de VIANA, 2010]**

O desenvolvedor de software sensível ao contexto que faz uso dessa definição deve determinar quais os elementos que compõem a zona de interesse em função dos objetivos do sistema. Assim, a zona de interesse de um sistema varia de um simples e resumido conjunto

de informações (e.g., o modelo do dispositivo de acesso ao sistema) a uma composição de elementos mais ricos semanticamente e mais complexo de capturar/inferir (e.g., o usuário caiu ao lado do telefone). Essas Zonas se alteram ao longo da execução do sistema em função da capacidade de observação e do próprio interesse do sistema, o que torna o contexto dinâmico (os valores das propriedades das entidades mudam) e evolutivo (as entidades que compõem o contexto são alteradas). Essa definição é a base do algoritmo de aquisição de contexto da plataforma de *middleware* LoCCAM.

### 2.1.1 Desafios no Desenvolvimento de Aplicações Sensíveis ao Contexto

O desenvolvimento de aplicações móveis e sensíveis ao contexto apresenta diversos problemas e desafios. Um exemplo de desafio é a heterogeneidade. Esta ocorre devido a grande variedade de dispositivos, plataformas de desenvolvimento e protocolos e interfaces de comunicação existentes, muitas vezes incompatíveis umas com as outras [LIMA, 2011].Isso leva à necessidade de uma maior portabilidade e interoperabilidade dos aplicativos para a redução dos esforços e gastos exigidos para o desenvolvimento destes. As limitações inerentes à natureza dos dispositivos móveis, como as limitações do consumo de recursos (e.g., a energia da bateria, a memória) e a ausência de perenidade na conexão do dispositivo com a Internet também tornam o processo de desenvolvimento mais árduo.

A heterogeneidade de plataformas de desenvolvimento é uma realidade no tocante à elaboração de softwares para dispositivos móveis. Mesmo tendo propósitos similares, as plataformas costumam se diferenciar em diversos aspectos, como por exemplo, a linguagem de programação utilizada ou mesmo a arquitetura interna, de modo que aplicações construídas especificamente para uma determinada plataforma de desenvolvimento sejam incompatíveis para as demais.

Essa incompatibilidade entre plataformas traz as mesmas dificuldades e custos adicionais existentes na problemática da fragmentação de software para o desenvolvimento de versões das aplicações que possam ser executados em cada plataforma. Assim, o tempo e o esforço gastos no desenvolvimento de etapas específicas de uma determinada plataforma que não podem ser reutilizadas para as demais acabam resultando na necessidade de uma maior portabilidade do software [LUCRÉDIO, 2009].

Outro fator que deve ser levado em consideração no desenvolvimento de aplicações móveis e sensíveis ao contexto é a interoperabilidade. Ela é um dos principais requisitos de

um sistema ubíquo [NIEMELÄ e LATVAKOSKI, 2004]. A interoperabilidade pode ser definida como sendo a capacidade do software em compreender as informações trocadas entre componentes e dispositivos e, utilizando estas, dispor novas informações para serem trocadas [NIEMELÄ e LATVAKOSKI, 2004].

Uma forma de garantir a interoperabilidade entre sistemas é a ideia de adaptar interfaces heterogêneas umas às outras ou restringir as interfaces entre componentes de maneira a torná-las tanto quanto possível similares em termos de sintaxe [COULOURIS *et al*, 2012]. Sua principal dificuldade consiste nas possíveis incompatibilidades de semântica e sintaxe existente entre as interfaces, devido à heterogeneidade [COULOURIS *et al*, 2012].

## 2.2 Plataformas de *Middleware*

Uma abordagem utilizada visando prover interoperabilidade e reduzir a complexidade do desenvolvimento de aplicações sensíveis ao contexto é a adoção de plataformas de *middleware*. Uma plataforma de *middleware* funciona como uma camada intermediária entre as aplicações e o sistema operacional no qual estas estão executando, agindo no sentido de uniformizar a sintaxe das interfaces entre estes [GRIGORAS, 2006].

### 2.2.1 *Middleware* para Desenvolvimento de Aplicações Móveis e Sensíveis ao Contexto

Niemelä e Latvakoski consideram um sistema ubíquo como dividido em três camadas principais: aplicações, infraestrutura do sistema e, intermediando-as, a plataforma de *middleware* [NIEMELÄ E LATVAKOSKI, 2004]. Uma plataforma de *middleware* voltado especificamente para sistemas ubíquos tem algumas diferenças em relação a uma plataforma de *middleware* tradicional. Isso decorre do fato de ela ter que resolver questões similares às das plataformas de *middleware* voltadas para sistemas distribuídos. Entre elas, pode-se mencionar a coordenação e comunicação entre componentes distribuídos e o suporte para a captura das informações de contexto [HENRICKSEN *et al*, 2005].

Entre as características importantes de uma plataforma de *middleware* para esse domínio pode-se citar [MAIA *et al*, 2013] [HENRICKSEN *et al*, 2005] [MAIA, ROCHA E ANDRADE, 2009]:

- **Escalabilidade:** Os componentes de aquisição e gerenciamento de contexto e os protocolos de comunicação devem executar bem mesmo em sistemas que possuem grande número de sensores ou atuadores.

- **Facilidade de implantação e configuração:** Os componentes de um sistema sensível a contexto devem ser facilmente implementados e configurados.
- **Rastreabilidade e controle:** Os estados dos componentes e o fluxo de informações entre estes devem ser abertos à inspeção, de modo a proporcionar um melhor controle deste pelo usuário.
- **Suporte à heterogeneidade:** Deve-se suportar uma grande variedade de interfaces de rede, linguagem de programação e dispositivos com os mais variados recursos de hardware.
- **Suporte à mobilidade:** Todos os componentes do sistema podem ser móveis e os protocolos de comunicação do sistema devem se adaptar a essa condição.
- **Suporte à privacidade:** O fluxo de informação de contexto entre os componentes distribuídos de um sistema sensível a contexto deve ser controlado de acordo com as necessidades de privacidade do usuário.
- **Tolerância à falha de componentes:** O sistema sensível a contexto deve continuar a execução mesmo que aconteçam desconexões e eventuais falhas nos componentes.
- **Baixo acoplamento:** A plataforma de *middleware* deve agir como um intermediário entre a aplicação e a camada que fornece as informações de contexto requeridas, sem que seja necessário que a aplicação acesse diretamente os sensores ou o sistema operacional.
- **Transparência:** O *middleware* deve permitir que as informações contextuais estejam disponíveis para as aplicações sem que estas necessitem de informações sobre como esses dados contextuais foram adquiridos.

### 2.2.2 Limitações e Desafios da Abordagem

O desenvolvimento de plataformas de *middleware* tem algumas dificuldades e entraves. Um deles é a complexidade inerente de se manter um único *middleware* para todas as entidades e componentes de um ambiente dinâmico e heterogêneo, como os como os que executam as aplicações ubíquas [MAIA, ROCHA E ANDRADE, 2009]. Outra dificuldade que pode ser mencionada é o fato de que, normalmente, as plataformas de *middleware* são baseadas em uma arquitetura cliente/servidor na qual quase nenhuma das tarefas de gerenciamento de contexto é executada diretamente no dispositivo móvel. Com isso, essas

plataformas comumente não conseguem adaptar os componentes com as quais elas capturam, processam e fornecem as informações de contexto [MAIA, ROCHA E ANDRADE, 2009].

Um exemplo de plataforma de *middleware* que busca solucionar alguns desses entraves é o LoCCAM [MAIA *et al.*, 2013], que será descrito na Seção a seguir.

## 2.3 LoCCAM

O LoCCAM é uma infraestrutura de gerenciamento de contexto voltada para dar suporte a aplicações sensíveis ao contexto em dispositivos móveis, intermediando de forma adaptativa a aquisição das informações contextuais. Tais informações contextuais podem ter como fonte os sensores físicos, virtuais e lógicos existentes no dispositivo móvel no qual está atuando [FONTELES, 2013].

O desenvolvimento do LoCCAM seguiu cinco princípios [MAIA *et al*, 2013]:

1. Os requisitos da aquisição de contexto foram desenvolvidos seguindo os conceitos do desenvolvimento de software baseado em componentes. No LoCCAM, os elementos responsáveis pela captura das informações contextuais são os CACs (Componentes de Aquisição de Contexto).
2. A informação contextual é disponibilizada através de espaço de tuplas ou notificações do tipo *publish-subscribe*. No LoCCAM, essa disponibilidade é possível através do SysSU (*System Support for Ubiquity*) [LIMA, 2011], que funciona como um intermediário da comunicação dos dados entre o *middleware* e as aplicações.
3. A camada de aquisição de contexto e as aplicações são desacopladas. No LoCCAM, quem permite esse desacoplamento é o SysSU, pelo já mencionado fato de intermediar a comunicação existente em ambas camadas.
4. A aquisição de informação contextual deve ocorrer de forma transparente. No LoCCAM, as informações contextuais estão disponíveis para as aplicações através de um espaço de tupla, de modo que as aplicações não precisam se preocupar com a origem destas informações para poder utilizá-las.
5. A adaptação da aquisição de contexto deve possibilitar a economia de recursos. A arquitetura do LoCCAM foi projetada especificamente para possibilitar a adição ou remoção de CACs durante o desenvolvimento, de acordo com as características da aplicação que se pretende construir e do dispositivo no qual se pretende esta aplicação

seja executada. Desta forma, o LoCCAM permite uma vasta gama de configurações possíveis.

### 2.3.1 Arquitetura

A Figura 2.2 apresenta a arquitetura que permitiu que o *middleware* pudesse seguir os cinco princípios mencionados na subseção anterior. O LoCCAM pode ser dividido em duas partes principais: a ferramenta SysSU e o framework CAM (*Context Acquisition Manager*) [FONTELES, 2013]. O SysSU atua como um intermediador. Ele recebe a relação de interesses das aplicações e as repassa para o CAM (mais especificamente, para o *Adaptation Reasoner*), assim como recebe as informações contextuais capturadas pelo CAM (mais especificamente, pelos CACs) e as disponibiliza para as aplicações.

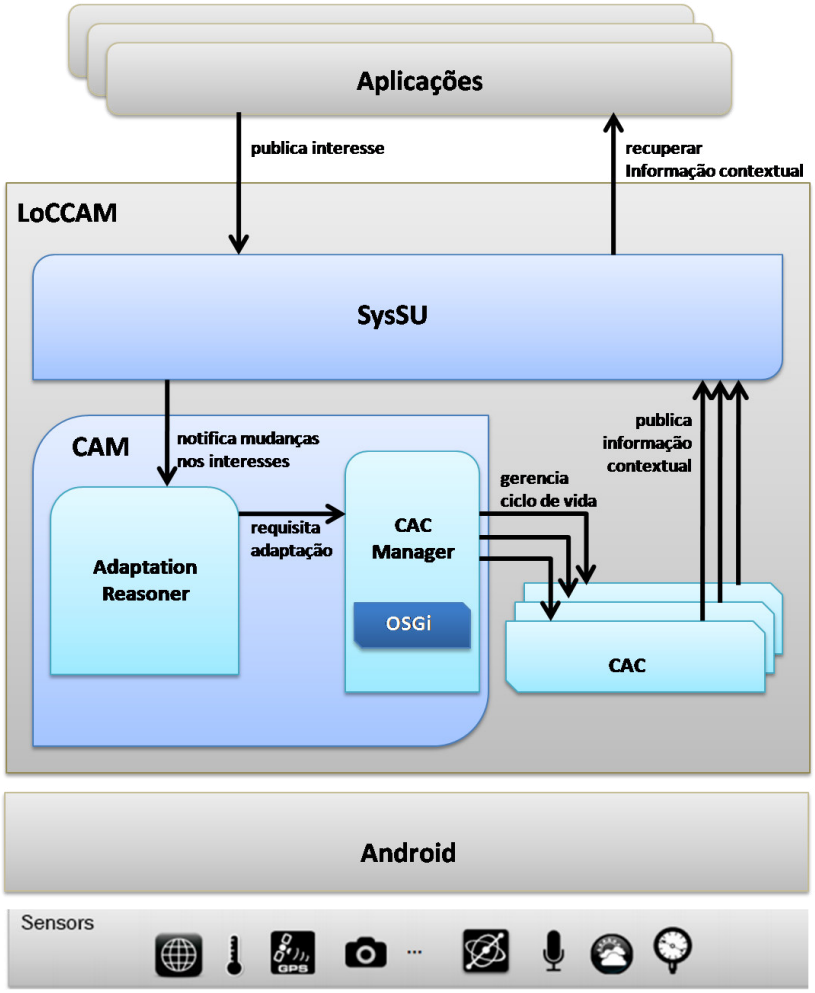


**Figura 2.2 – Arquitetura do LoCCAM [Adaptado de FONTELES, 2013]**

O SysSU é uma ferramenta que age como um sistema de suporte cujo modelo é um misto dos modelos Linda [CARRIERO E GELERNTER, 1989] e *publish-subscribe* [LIMA,

2011]. A sua principal vantagem é possibilitar a interoperabilidade do sistema e permitir o desacoplamento entre as camadas. O SysSU será melhor descrito na subseção 2.3.1.1.

O *framework* CAM pode ser dividido em sua própria arquitetura interna: o *Adaptation Reasoner* é o responsável por manter uma lista com a relação de interesses de todas as aplicações e por verificar eventuais modificações que possam ocorrer nessa relação de interesse.O *CAC Manager* é o responsável pelo gerenciamento do ciclo de vida dos CACs [FONTELES, 2013]. O CAM será mais bem descrito na subseção 2.3.1.2.

Os CACs são os responsáveis pela aquisição das informações contextuais requeridas pelas aplicações e pela publicação destas no SysSU.

### 2.3.1.1 SysSU

O SysSU é uma infraestrutura que funciona como sistema de suporte para aplicações móveis e sensíveis ao contexto [LIMA, 2011]. Ele funciona como uma camada de armazenamento das informações contextuais, disponibilizadas na forma de tuplas e podendo ser acessadas de forma síncrona ou assíncrona [FONTELES, 2013].

A arquitetura do SysSU é apresentada na Figura 2.3. Originalmente, ela funcionava como cliente/servidor. Dessa maneira, o *UbiCentre* ficava em servidor, enquanto o *UbiBroker* ficava em cada dispositivo-cliente. O *UbiBroker* é o *middleware* que realiza a comunicação entre os dispositivos e o *UbiCentre*, que funcionava como um repositório de espaço de tuplas. O *UbiCentre* disponibiliza as informações contextuais para vários *UbiBrokers* que o acessam concorrentemente [LIMA, 2011].

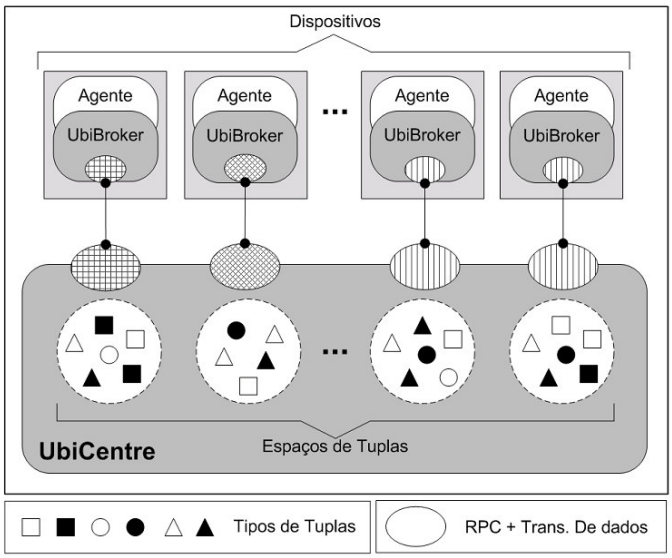


**Figura 2.3 – Arquitetura do SysSU [LIMA et al, 2011]**

O modelo original do SysSU foi adaptado para ser integrado ao LoCCAM e prover o desacoplamento entre as aplicações e camada de aquisição de contexto. Ele está totalmente embarcado no dispositivo móvel junto com LoCCAM e funciona como um serviço no Android [FONTELES, 2013].

#### 2.3.1.1.1 Filtros Contextuais

Os filtros contextuais do SysSU são uma característica desta infraestrutura que é utilizada no LoCCAM. Através desses filtros, é permitida a busca de informações contextuais específicas dentro do espaço de tuplas. Com eles, é possível realizar consultas buscando informações contextuais que satisfaçam alguma condição. Por exemplo, saber se a temperatura ambiente está em uma faixa de valor específica. Os filtros também podem ser desenvolvidos visando à realização de consultas internas, buscando dados de outras informações contextuais no intuito de se poder inferir uma condição que necessite da combinação dessas informações contextuais. A Figura 2.4 mostra um exemplo do que pode ser realizado com um filtro. No caso demonstrado, o filtro retorna para a aplicação apenas as tuplas que representem temperaturas cujo valor exceda 30 graus.

```
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 2.4 – Exemplo de Filtro Contextual [FONTELES, 2013]**

Originalmente, os filtros do SysSU foram criados para serem implementados em Javascript. No entanto, para o LoCCAM, os filtros foram adaptados para serem implementados em Java. Uma limitação existente nos filtros do SysSU reside na complexidade do código para descrever condições de "eventos". Dessa forma, o filtro mostrado na Figura 2.4 continuaria a notificar a aplicação enquanto a temperatura excedesse 30 graus Celsius. Para conseguir descrever uma condição em que a aplicação é notificada apenas quando a temperatura exceder 30 graus Celsius, faz-se necessário o uso de dois filtros.

O segundo filtro seria uma negação do primeiro e poderia “desativar” o primeiro filtro no instante em que a condição desejada é atingida e o “reativar” quando esta deixa de ser satisfeita.

#### 2.3.1.1.2 Subscrições

Outra possibilidade de acesso das informações contextuais provida pelo SysSU são as subscrições. Através delas, é possível notificar uma aplicação toda vez que o valor de uma informação contextual sofrer uma motivação no SysSU. Uma subscrição comum do LoCCAM requer que o desenvolvedor comunique a informação contextual que ele deseja subscrever; um filtro contextual, que define a condição para que a aplicação seja notificada e um *reaction*, que descreve o que a aplicação deve fazer a condição descrita no filtro é satisfeita. A Figura 2.5 apresenta um exemplo de subscrição implementada no LoCCAM. No caso exemplificado, a subscrição recebe uma informação contextual chamada “Random”, que publica a cada 10 segundos um valor numérico aleatório. Sempre que o valor publicado por Random for maior do que “10”, a aplicação é notificada pela subscrição e, quando isso acontece, o *reaction* publica o valor da informação contextual recebida no *console*.

```
Pattern p = (Pattern) new Pattern()
    .addField("ContextKey", "context.random");
try {
    reactionId = service.subscribe(new IClientReaction.Stub() {
        public void react(Tuple tuple) throws RemoteException {
            System.out.println("REACT: " tuple.getField(2).getValue().toString());
        }
    }, "put", p, new IFilter.Stub() {
        public boolean filter(Tuple tuple) throws RemoteException {
            for(int i = 0; i < tuple.size(); i++) {
                if(tuple.getField(i).getName().equalsIgnoreCase("Values")) {
                    List values = (List)tuple.getField(i).getValue();

                    int v = Integer.parseInt(values.get(0));

                    if(v>10)
                        return true;
                }
            }
            return false;
        }
    });
} catch (RemoteException e) {
    Toast.makeText(SysSUClientTestActivity.this, "Erro na subscrição",
                                                    Toast.LENGTH_LONG).show();
    e.printStackTrace();
}
```

**Figura 2.5 – Exemplo de Subscrição**

### 2.3.2 Componentes de Aquisição de Contexto

Os CACs são a unidade básica do *middleware* LoCCAM, sendo os responsáveis pela aquisição das informações contextuais. Cada CAC encapsula um sensor. Este sensor pode ser tanto um sensor físico (e.g., acelerômetro), quanto um sensor lógico (e.g., a inferência de uma

informação contextual através da combinação das informações obtidas por diversos sensores) ou um sensor virtual (e.g., um serviço *web* que forneça temperatura da cidade ou outros dados meteorológicos) [MAIA *et al*, 2013] [FONTELES, 2013].

O ciclo de vida de um CAC é gerenciado pelo *CACManager* e é composto de três estados: "ativo", "instalado" e "desinstalado".

O estado "desinstalado" é o estado padrão do CAC, no momento em que ele é adicionado ao repositório de CACs do dispositivo móvel. Quando o *CACManager* é informado da adição de um novo componente, ele modifica o estado do CAC para "instalado", deixando-o disponível na lista dos componentes que podem ser ativados ou não, dependendo dos interesses das aplicações [FONTELES, 2013].

Quando uma aplicação publica o interesse na informação correspondente a um CAC instalado, o *CACManager* recebe este dado e modifica o estado do CAC para "ativo". No estado "ativo", o CAC está execução e pode adquirir as informações contextuais e publicá-las no SysSU [FONTELES, 2013]. Quando nenhuma aplicação necessita mais da informação contextual fornecida pelo CAC, o *CACManager* o coloca novamente no estado "instalado".

### 2.3.3 Representação do Modelo de Contexto

Para que os CACs e as aplicações pudessem se comunicar é necessária a existência de um vocabulário comum. Este tem que permitir que, através do SysSU, a aplicação possa comunicar qual a informação contextual que tem interesse e o CAC qual a informação que está publicando. Para possibilitar tal comunicação foi desenvolvido o conceito de *Context Keys*. Elas funcionam como um vocabulário compartilhado que representa cada tipo de informação contextual. Assim, cada *Context Key* provê um nome único que deve ser utilizado pelo CAC para determinar qual informação contextual ele publica.

Para gerar as *Context Keys* de maneira única, elas foram definidas através de uma árvore de hierarquia. Esta árvore de hierarquia, mostrada na Figura 2.6, é inspirada no modelo *Management Information Base* (MIB)[1]. Cada *Context Key* é formada pela sequência de nomes dos nós separados por pontos. Por exemplo, uma *Context Key* que serve para descrever a temperatura ambiente é "context.ambient.temperature". Outro exemplo de *Context Key* que representa o nome do usuário da aplicação é "context.user.name".

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

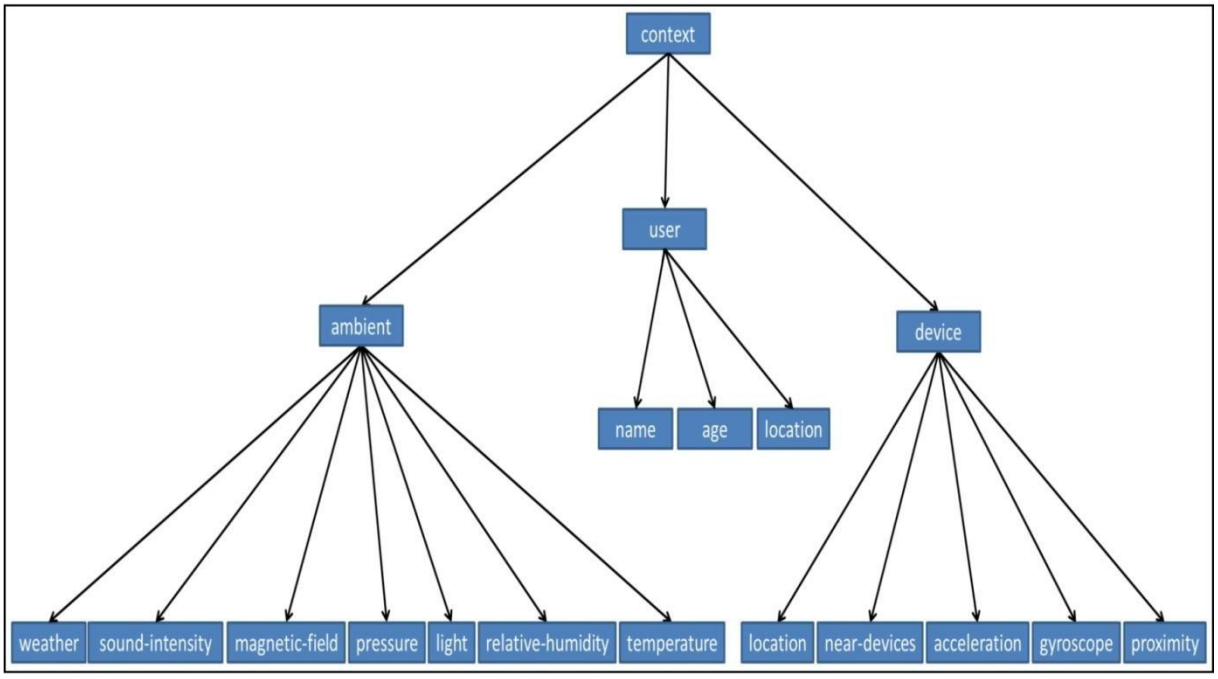


**Figura 2.6– Árvore de Hierarquia Antiga [FONTELES, 2013]**

## 2.4 Engenharia Dirigida por Modelos

Em vista do crescente aumento na complexidade no desenvolvimento de softwares e a incapacidade de linguagens como Java ou C# em lidar com tal complexidade e expressarem os conceitos de domínio adequadamente, surgiu a MDE (*Model-Driven Engineering*) [SCHMIDT, 2006], uma nova abordagem que passa a tratar a modelagem como a principal etapa no desenvolvimento.

Segundo Schmidt, a MDE trás a combinação de dois elementos primordiais: a modelagem em alto nível, realizada utilizando linguagens de modelagem de domínio específico, e a transformação, possibilitada pelo desenvolvimento de *engines* próprias para tal e geradores de código [SCHMIDT, 2006]. Neste caso específico, "*engines*" são ferramentas desenvolvidas com o objetivo de maximizar a automação do processo.

### 2.4.1 Visão Geral

A Figura 2.7 apresenta uma visão geral, com os principais elementos e etapas presentes em um processo de desenvolvimento de software que segue a abordagem da MDE.

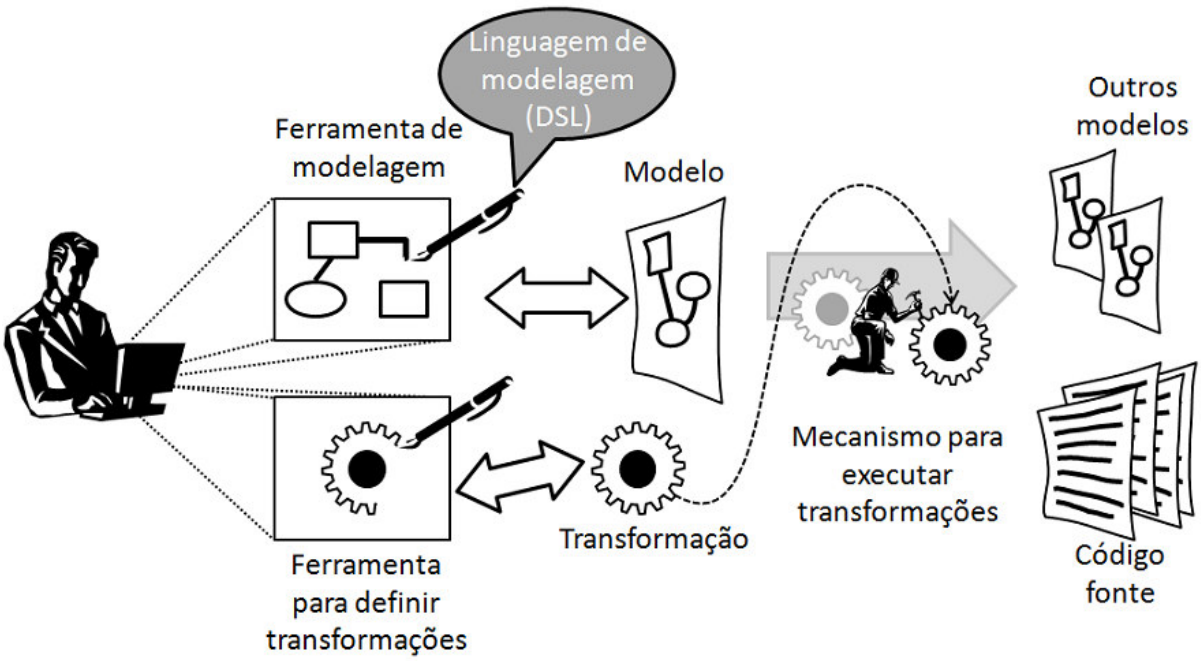


**Figura 2.7– Principais Elementos da MDE [LUCRÉDIO, 2009]**

A equipe de desenvolvimento se foca em duas frentes: a construção dos modelos, de acordo com o domínio do problema e os requisitos do sistema. Para tal, eles utilizam as Linguagens Específicas de Domínio (DSL) e a definição das regras de transformação que envolvem o modelo construído. Uma vez que o modelo e as regras estejam prontos, uma *engine* combina ambos e realiza a transformação deste modelo inicial em um código-fonte ou em um novo modelo mais específico. Esse modelo gerado então passa por novas etapas de modelagem e transformação até que o produto final gerado seja o código-fonte esperado pela equipe de desenvolvimento.

#### 2.4.1.1 Vantagens

A utilização do paradigma do MDE trás diversas vantagens para o processo de produção de um software. Uma breve lista com os principais benefícios existentes no uso da MDE pela equipe de desenvolvedores é apresentada abaixo [LUCRÉDIO, 2009]:

- A constante atualização de modelos e da documentação. Esta é decorrente do conceito de tornar os modelos parte constituinte do software. Assim, quaisquer modificações que devam ser feitas no software são feitas diretamente no nível do modelo.
- A agilidade na manutenção. O uso do paradigma MDE auxilia a equipe de desenvolvedores a encontrar as correções que devem ser realizadas, enquanto a

automatização das transformações diminui o tempo necessário para levar tais modificações conceituais ao nível do código.

- Maior facilidade de reuso, portabilidade e interoperabilidade. Na MDE, é possível de um único modelo, de nível de abstração mais alto, gerar vários modelos específicos, para as mais diferentes plataformas, resultando em códigos implementados para tais plataformas.
- A eliminação da possibilidade de falhas humanas durante a codificação. Com a MDE, são eliminadas chances de erros devido à má-interpretação de um trecho do modelo, falta de atenção ou por desconhecimento, por parte do desenvolvedor, de certas propriedades da linguagem ou plataforma com a qual se está trabalhando.
- A otimização da produtividade. Este é um benefício que é resultante de todas as demais vantagens descritas.

#### 2.4.1.2 Desvantagens

Dentre as principais desvantagens e limitações provenientes da utilização do paradigma MDE, se pode citar [LUCRÉDIO, 2009]:

- A rigidez do código gerado. Ela é causada pela automatização da transformação. Os códigos gerados pelas ferramentas de transformação são padronizados, em uma analogia semelhante aos blocos de montar.
- A impossibilidade de otimização diretamente no código-fonte. Embora, na prática, o desenvolvedor possa mexer no código, isto fere os princípios da MDE ao afastar o código implementado real do modelo desenvolvido
- O desempenho prejudicado. Isto é resultado direto da rigidez do código e faz com que uma aplicação desenvolvida utilizando os conceitos da MDE possa vir a ter um desempenho inferior se comparada com uma equivalente que tenha sido desenvolvida diretamente no código-fonte e devidamente otimizada.
- Um possível alto investimento inicial no treinamento da equipe de desenvolvedores. Dada a possibilidade de a equipe de desenvolvedores não esteja habituada à utilização da MDE, pode-se ter custo inicial referente ao treinamento desta e da construção de uma infraestrutura que possibilite o uso desse paradigma.

### 2.4.1.3 Abordagens Existentes

O desenvolvimento das abordagens da MDE não teve uma evolução unificada. Diversos projetos e iniciativas progrediram simultaneamente, competindo entre si e criando diversas ferramentas e tecnologias, cada uma voltada para as suas abordagens e visões dos conceitos desse paradigma. Muitas dessas abordagens foram tanto de âmbito industrial como acadêmicos.

Dentre as principais abordagens existentes atualmente para a MDE, pode-se citar a MDA (cujo conjunto de padronizações desenvolvidas pela OMG) e as abordagens criando dentro do projeto Eclipse, embora existam vários outros projetos e abordagens menores independentes.

#### 2.4.1.3.1 Abordagem MDA

A MDA é um conjunto de padrões e conceitos desenvolvidos com a ideia de nortear todas as fases do desenvolvimento de softwares orientados a modelos. Dentre os principais benefícios esperados pela utilização da MDA estão a otimização da produtividade, da interoperabilidade e da portabilidade [KLEPPE, WARMER E BAST, 2003].

Entre os padrões adotados pela MDA, pode-se citar:

- **MOF (*Meta-Object Facility*):** É um meta-metamodelo através do qual são criados os metamodelos de acordo com a abordagem MDA.
- **UML (*Unified Modeling Language*):** Adotada para a modelagem e representação visual dos modelos.
- **QVT (*Queries/Views/Transformations*):** É um padrão de linguagens voltadas para transformação do modelo.
- **XMI (*XML Metadata Interchange*):** É um padrão desenvolvido para a representação dos modelos no formato XML.

Outro conceito importante apresentado pela MDA são os três níveis de abstração que um modelo pode ter e as relações existentes entre eles. São eles:

- **CIM (*Computation Independent Model*):** O modelo independente de computação, que seria o equivalente ao nível mais alto de abstração, equivalente aos requisitos do sistema ou ao modelo de domínio deste.

- **PIM (*Platform Independent Model*):** Modelo independente de plataforma, que seria um nível de abstração do modelo em que ele já adiciona conceitos de computação, mas ainda em um grau intermediário de modo a torná-lo um denominador comum a quaisquer plataformas de implementação.
- **PSM (*Platform Specific Model*):** Modelo específico de plataforma, que seria um nível de abstração mais baixo, no qual o modelo já seria projetado de acordo com as características e especificidades de cada plataforma.

As transformações acontecem entre um nível e outro de abstração, tornando-o mais detalhado. Por suas características, um modelo PIM pode ser transformado em n modelos PSM equivalentes. Estes, por sua vez, seriam o estágio final para ser transformados em código-fonte. A Figura 2.8 mostra a relação de transformações existente entre CIM, PIM e PSM.

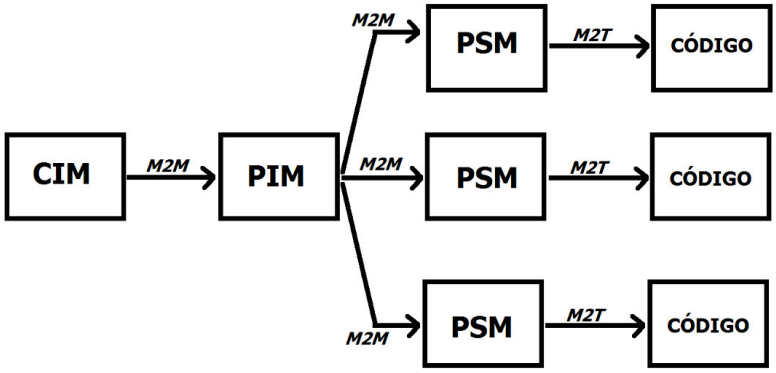


**Figura 2.8– Relação existente entre CIM, PIM, PSM**

## 2.4.1.3.2 Abordagem Eclipse

A abordagem Eclipse para a MDE corresponde a um grupo de projetos, *frameworks* e ferramentas desenvolvidas para a plataforma Eclipse[2]. Originalmente, o desenvolvimento de aplicações relacionadas à MDE dentro do Projeto Eclipse seguia a padronização MDA e seu principal carro-chefe, o *framework* EMF, utilizava uma versão do MOF como seu meta-metamodelo. No entanto, por decisões de projeto, o Eclipse decidiu abandonar o padrão MDA em prol de uma abordagem própria [LUCRÉDIO, BITTAR E FORTES, 2009]. O núcleo da abordagem Eclipse é o meta-metamodelo Ecore, em contraposição ao MOF, que é utilizado por todos os projetos desta abordagem, em especial o EMF.

O EMF (Eclipse Modeling Framework) é um *framework* do Eclipse que permite a

2 http://eclipse.org/

criação e manipulação de meta-modelos e modelos. Desenvolvido em Java, ele permite a manipulação de modelos seguindo como base um metamodelo criado através do próprio *framework*. Com base nesses modelos, é possível desenvolver ferramentas e outros tipos de aplicação [ECLIPSE PROJECT, 2005].

Embora o EMF seja o principal *framework* da abordagem Eclipse, esta também abriga um grande número de projetos e ferramentas de desenvolvimento voltadas para a MDE, correlacionadas ou independentes. Algumas destas são citadas a seguir.

- **GMF (*Graphical Modeling Framework*)[3]:** Este *framework* é uma ferramenta voltada para a criação e manipulação de linguagens visuais de domínio específico. Ele objetiva permitir a definição completa de um ambiente de modelagem para uma linguagem visual específica para um determinado domínio [LUCRÉDIO, 2009]. Para conseguir tal definição, é necessário que, primeiramente, se faça a definição do metamodelo dessa linguagem visual, com base no meta metamodelo Ecore. Com base no metamodelo criado, é possível definir a aparência gráfica dos elementos visuais dessa linguagem e as ferramentas necessárias para a criação dos elementos. Através da combinação dessas três definições é gerado o ambiente de modelagem, na forma de um *plugin* para o Eclipse.
- **JET (*Java Emitter Templates*)[4]:** O JET é uma ferramenta de geração de códigos baseada em *templates* [LUCRÉDIO, 2009]. O JET trabalha utilizando códigos Java e algumas marcações pré-determinadas dentro dos *templates*, além de poder receber os modelos EMF como entrada, servindo assim, como gerador de código.
- **GMT (*Generative Modeling Tools*)[5]**: O GMT é um subprojeto do Eclipse que serve como uma incubadora para projetos de ferramentas, linguagens e *frameworks* voltados para os mais diversos aspectos envolvendo os conceitos da MDE. Tais projetos apresentam diferentes níveis de maturidade. De forma que, alguns se desprenderam do GMT e tornaram-se um projeto ou subprojeto em si [LUCRÉDIO, 2009] [STAHL e VÖLTER, 2006].Dentre os subprojetos que fazem parte do GMT, podem-se citar:
    - **ATL (*Atlas Transformation Language*)[6]:** É uma linguagem de transformação do tipo M2M. Embora tenha diversas equivalências com a linguagem QVT, o

3 http://www.eclipse.org/modeling/gmp/
4 http://www.eclipse.org/modeling/m2t/?project=jet
5 http://www.eclipse.org/gmt/
6 http://www.eclipse.org/atl/

ATL tem como base o Ecore. Por ser uma linguagem que faz parte da abordagem Eclipse, o ATL tem *plugin* para o desenvolvimento de transformação no Eclipse e trabalha integrado ao EMF [STAHL e VÖLTER 2006].

- **MOFScript[7]:** É uma linguagem que visa a transformação de modelos do tipo M2T) e que trás compatibilidade com a linguagem-padrão QVT [LUCRÉDIO, 2009].
- **TCS (*Textual Concrete Syntax*)[8]:** É uma ferramenta que permite a especificação da sintaxe concreta de DSLs textuais. É voltada para facilitar o desenvolvimento de transformações M2T em DSLs textuais [LUCRÉDIO, 2009] [JOUAULT, BÉZIVIN e KURTEV, 2006].

- ***openArchitectureWare*[9]:** Era um antigo projeto incubado do GMT que se desvinculou. Ele engloba o desenvolvimento de ferramentas e linguagens para a modelagem de DSLs textuais e geração de código através de *templates* [LUCRÉDIO, 2009]. Dentre as suas principais ferramentas, pode-se citar:
  - **xText[10]:** É uma linguagem para modelagem de DSLs textuais, permitindo especificar a sintaxe concreta destas linguagens a partir de um metamodelo criado em EMF.
  - **xPand[11]:** É uma linguagem que a criação de *templates* que serão utilizados na geração de códigos. Pode atuar tanto em conjunto com o MWE quanto ser chamado programaticamente pelo Java.
  - **MWE (*Modeling Workflow Engine*)[12]:** É um *engine* baseada em XML (ou em uma notação similar à Java no MWE2) que permite a execução de uma transformação de geração de código, tendo como base *templates* xPand ou modelos Xtext.

Neste trabalho, foram adotados o *framework* GMF para a modelagem visual das informações contextuais e a linguagem xPand para a implementação dos *templates* das regras

7 http://www.eclipse.org/gmt/mofscript/
8 http://www.eclipse.org/gmt/tcs/
9 http://www.eclipse.org/gmt/oaw/
10 http://www.eclipse.org/Xtext/
11 http://www.eclipse.org/modeling/m2t/?project=xpand
12 http://www.eclipse.org/modeling/emft/?project=mwe

de transformação.

### 2.4.2 Linguagens de Domínio Específico

As linguagens de domínio específico (do inglês, *Domain-Specific Language*, cuja sigla é DSL) são a base nas quais os modelos construídos são fundamentados. Segundo [DEURSEN, KLINT, E VISSER, 2000], as DSLs são linguagens com poucas palavras reservadas, normalmente declarativas, e cujo foco está em um domínio de problema específico, oferecendo notações e abstrações apropriadas para tal.

As DSL podem tanto ser textuais quanto visuais, apresentando algumas diferenças em ambos os tipos, que vão desde a forma como os conceitos do domínio são criados e representados, até a forma como é realizada a representação visual da DSL para os usuários. Comparativamente, DSLs visuais são mais intuitivas que as textuais, além de apresentarem uma maior facilidade de uso por leigos [LUCRÉDIO, BITTAR E FORTES, 2009].

Para se desenvolver uma DSL são necessárias três etapas fundamentais [FEIKAS, 2006]:

- **Definição da sintaxe abstrata:** Esta compreende os conceitos do domínio, bem as relações e restrições que devem ser aplicadas a esses conceitos [LUCRÉDIO, BITTAR E FORTES, 2009]. A maior parte das DSL costuma utilizar um meta-modelo como forma de sintaxe abstrata [FEIKAS, 2006].
- **Definição da sintaxe concreta:** Uma vez que a sintaxe abstrata tenha sido definida, é necessária uma sintaxe que represente grafica ou textualmente os conceitos expressos por esta. No caso de DSL visual, os conceitos seriam representados através de ícones gráficos, enquanto em uma DSL textual, estas seriam definidas por uma gramática [LUCRÉDIO, BITTAR E FORTES, 2009].
- **Definição da semântica:** É a definição do significado dos conceitos especificados na sintaxe abstrata. A semântica pode ser um conjunto de ações, descrições ou mesmo uma ferramenta que sirvam para traduzir os elementos conceituais da DSL para o que se deseja expressar através desta [LUCRÉDIO, BITTAR E FORTES, 2009].

### 2.4.3 Transformação

A transformação pode ser definida como o processo de gerar, de forma automática (ou semi), um código-fonte ou um modelo mais específico a partir de um modelo previamente definido.

### 2.4.3.1 Tipos de Transformação

As transformações que podem ser realizadas a partir de um modelo podem ser classificadas em duas categorias principais [LUCRÉDIO, 2009]:

- **M2M (*model-to-model*):** Trata-se da transformação entre um modelo com maior grau de abstração para um modelo equivalente com maior grau de especificação. Usualmente, essas transformações são utilizadas como etapas intermediárias no processo do desenvolvimento de um software. Um exemplo que pode ser mencionado de utilização das transformações M2M como etapa intermediária são as transformações que ocorrem nas primeiras etapas de um processo que siga o padrão MDA. A MDA faz uma distinção dos modelos em três níveis de abstração. Ela parte do mais geral, o modelo CIM, que representa os modelos no mais alto grau de abstração e mais próximo do domínio do problema do que da solução, até o mais específico, o modelo PSM, que representa os modelos no mais alto grau de especificação, onde estes já incluem detalhes da plataforma para a qual ocorrerá a transformação para código. Todas as transformações que levam de um modelo CIM até um modelo PSM equivalente são exemplos de transformações do tipo M2M.
- **M2T (*model-to-text*):** Trata-se da transformação em que, a partir de um modelo, é gerado um código-fonte. É o tipo de transformação mais comum e, algumas vezes, é denominado M2C (*model-to-code*). Um exemplo de utilização das transformações M2T também pode ser encontrado no processo que siga o padrão MDA. A última etapa de transformação envolve o nível de modelo mais específico, PSM. Através desse modelo deve ser gerado um código correspondente à plataforma para a qual ele foi especificado. Outro exemplo de utilização de M2T é a geração de código baseado em *templates*, onde o código-fonte é originado de acordo com os parâmetros de um *template* previamente existente.

### 2.4.3.2 Geração de Código Baseado em *Templates*

Um tipo de transformação M2T bastante comum é a transformação em que ocorre geração de código baseada em *templates*. Nesse caso, um *template* é um arquivo de texto que contenha construções para seleção e expansão de código. Essas construções realizam consultas em arquivos de entrada e utilizam os resultados como parâmetro na geração de um

código em linguagem textual. Desta forma, os *templates* são utilizados como definidores de regras de transformação de modelos em códigos [LUCRÉDIO 2009], tal como mostrado na Figura 2.9. Exemplos de linguagens e ferramentas que realizam a geração de código baseada em *templates* são a JET e o xPand.

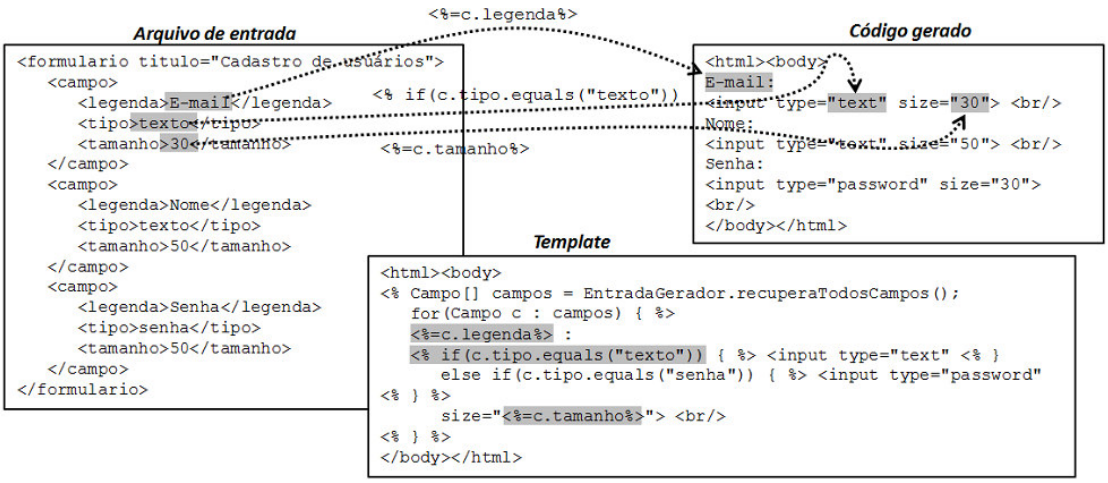


**Figura 2.9– Geração Baseada em Templates [LUCRÉDIO, 2009]**

## 2.5 Conclusão

Este capítulo introduziu os conceitos básicos de Sensibilidade ao Contexto, Plataformas de *Middleware* para Sistemas Sensíveis ao Contexto e Engenharia Dirigida por Modelos.

Com respeito à Sensibilidade ao Contexto, esta foi introduzida dentro do conceito de computação ubíqua imaginado por Mark Weiser e sua importância para sistemas ubíquos foi explicada. As definições sobre Contexto foram apresentadas, além de uma explicação sobre aplicações móveis e sensíveis ao contexto. Também foram detalhados alguns principais desafios e entraves no desenvolvimento de aplicações móveis e sensíveis ao contexto.

No que se refere à Plataforma de *Middleware*, estas foram definidas e explicadas quais as principais características que uma plataforma de *middleware* voltada para o desenvolvimento de aplicações sensíveis ao contexto deve ter e quais são as limitações desta abordagem. Dentro deste tema, também foi apresentado o *middleware* LoCCAM, explicando os seus princípios e os componentes fundamentais de sua arquitetura interna.

No tocante à Engenharia Dirigida por Modelos, foi apresentada uma visão geral de seus conceitos e como eles se relacionam. Além de uma discussão sobre as possíveis vantagens e desvantagens na utilização deste paradigma para o desenvolvimento de aplicações

móveis e sensíveis ao contexto. Foram apresentados os conceitos de Linguagens de Domínio Específico e Transformação, especificando os tipos de transformações existentes e uma das principais abordagens de geração de código utilizada.

A fundamentação teórica apresentada neste capítulo provê o embasamento necessário para o desenvolvimento deste trabalho, mostrando os conceitos primordiais a respeito de sensibilidade ao contexto, *model-driven enginnering* e plataformas de *middleware*. O próximo capítulo apresenta os trabalhos relacionados que envolvem os três temas conjuntamente visando o desenvolvimento de aplicações móveis e sensíveis ao contexto.

# Capítulo 3

# Trabalhos Relacionados

Neste capítulo, são apresentados os principais trabalhos encontrados relacionados ao conceito de DSL construídas enfocando o desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizem plataformas de *middleware* para a aquisição contextual. Além disso, é apresentado um conjunto de características desejáveis para DSLs desta categoria.

Na Seção 3.1, são apresentados os trabalhos relacionados ao tema desta dissertação. Na Seção 3.2,são apresentadas as características elicitadas durante o estudo dos trabalhos e é realizado um comparativo entre as características levantadas e as abordagens existentes na Literatura, de modo a verificar quais destas satisfazem ou não as características.

## 3.1 DSLs voltadas para Plataformas de Middleware Sensíveis ao Contexto

Nesta Subseção, são apresentados os trabalhos que apresentam DSLs voltadas ao desenvolvimento de aplicações sensíveis ao contexto com base em uma plataforma de *middleware* que fornece as informações contextuais. Entre as informações mostradas estão o modelo de contexto adotado, a forma como são representadas as informações contextuais e os elementos contextuais em cada linguagem, além da finalidade delas e para qual *middleware* a linguagem foi desenvolvida, caso ela seja específica para apenas uma plataforma de *middleware*.

Para selecionar os trabalhos, foi realizada uma investigação em *engines* de busca (e.g., Google Acadêmico, IEEE, ACM) em busca de DSLs para o domínio de aplicações sensíveis ao contexto. Do resultado desta busca, foi realizada uma triagem para selecionar obras que combinassem DSLs e plataformas de *middleware* para o desenvolvimento de aplicações sensíveis ao contexto. Foram encontrados e selecionados na literatura cinco trabalhos com abordagens voltadas para esse nicho de linguagens em específico. São estes:

- A abordagem de Almeida *et al* [ALMEIDA *et al*, 2005] [ALMEIDA *et al*, 2006];
- A abordagem de Daniele *et al* [2009];

- Copal-DSL [LI, SEHIC e DUSTDAR, 2010] [SEHIC, LI e DUSTDAR, 2011];
- MLContext [HOYOS, GARCÍA-MOLINA e BOTÍA, 2010][HOYOS, GARCÍA-MOLINA e BOTÍA, 2013];
- A abordagem de Santos *et al* [SANTOS *et al*, 2011][SANTOS *et al*, 2013].

### 3.1.1 Abordagem de Almeida *et al*

A DSL apresentada por Almeida *et al* [ALMEIDA *et al*, 2005] tem como característica principal a divisão em três níveis de abstração e independência de plataforma: o *service specification level*, onde o comportamento do serviço sensível ao contexto é descrito abstraindo-se com a informação contextual é obtida; o *platform-independent service design level*, em que o comportamento do serviço sensível ao contexto é descrito como um serviço da plataforma A-MUSE [TAKAHASHI, SUGANUMA e SHIRATORI, 2005], definindo como os serviços de contexto e ações são registrados, buscados e usados pelos *coordination component*; e o *platform-specific service design level*, que descreve a realização do serviço em si em uma plataforma particular. Esses três níveis são mostrados na Figura 3.1.

O serviço sensível ao contexto descrito pela aplicação consiste basicamente do *context sources*, que proveem a informação contextual para os *coordination components*, que, quando necessário, requerem as ações para serem executadas pelos *action providers*. A interface entre o usuário e os *coordination components* é o *user components* [ALMEIDA *et al*, 2005].

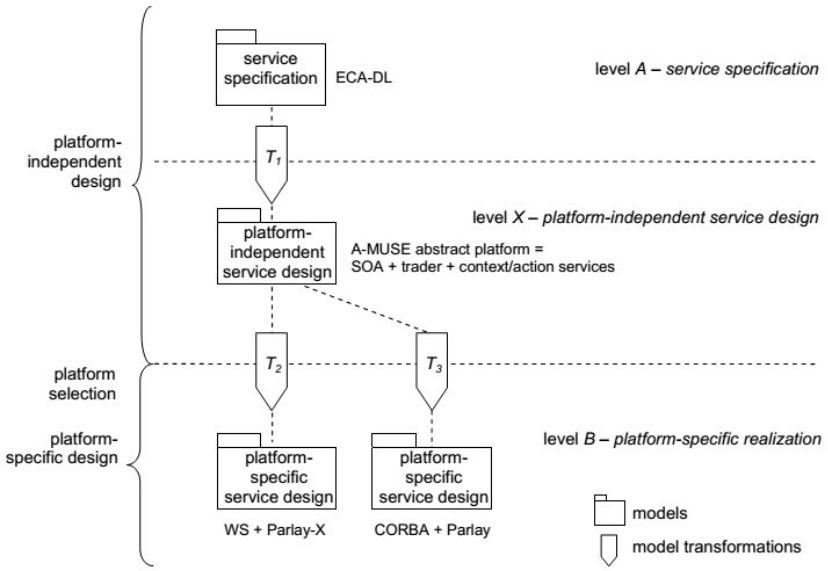


**Figura 3.1 – Os três níveis de modelos da abordagem de Almeida *et al* [2006]**

No primeiro nível de abstração, a linguagem utilizada foi nomeada como ECA-DL (*Events-Conditions-Actions Domain Language*), uma especialização dos elementos de uma

linguagem de propósito geral, a ISDL (*Interaction System Design Language*) [DIRGAHAYU, 2005]. O serviço que está sendo modelado é descrito em termos de *events*, *queries* e *actions*. *Events* representam mudanças contextuais; *queries* representam fontes contextuais; e *actions* representam as ações [ALMEIDA *et al*, 2007]. Com esta linguagem, é possível especificar os comportamentos em termos de ações e relações de casualidade entre estas ações. No ECA-DL, os atributos das informações contextuais são representados visualmente através de diagramas de classe UML, embora as restrições sejam expressas textualmente, de forma que seja correspondentes diretamente à expressões OCL (*Object Constraint Language*). A Figura 3.2 apresenta um exemplo de modelo expresso em ECA-DL. No exemplo apresentado, o evento seizureAlert_ind recebe uma informação contextual do tipo "Pacient" e invoca uma ação chamada alertTeam_reqA. Essa ação aciona um alerta para o Paciente do tipo "Ataque Epilético". O tipo "Paciente" é definido através de uma classe UML.

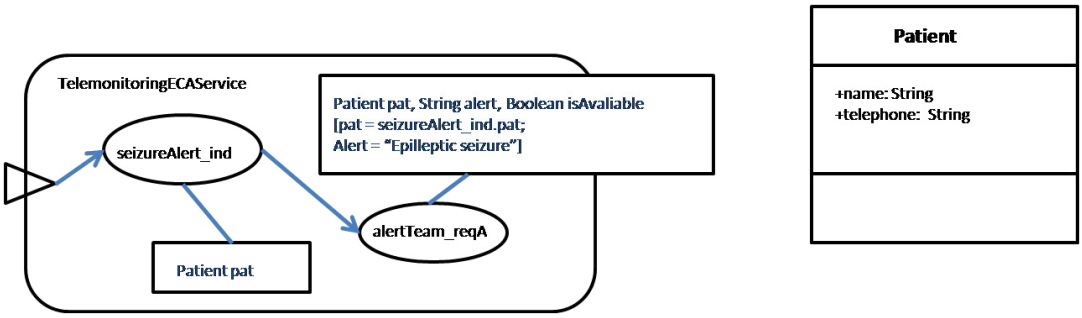


**Figura 3.2 – Exemplo de modelo no nível do ECA-DL [ALMEIDA *et al*, 2005]**

A partir do segundo nível de abstração, a linguagem utilizada passa a ser a IDSL. As informações e atributos de localização das ações são descritos em UML, com as restrições de tais atributos descritas em OCL.

Ocorrem duas transformações no modelo descrito por Almeida *et al*: do primeiro nível de abstração para o segundo, o modelo ECA-DL é transformado em um modelo ISDL utilizando a ferramenta de transformação GReAT (*Graph Rewriting And Transformation*). A segunda transformação resulta em uma especificação do serviço de contexto específico de uma determinada plataforma, como BPEL (*Business Process Execution Language*)[13] ou *Web Service*.

Entre as vantagens da abordagem, pode-se mencionar o uso de vários níveis de abstração, o que auxilia no processo de desenvolvimento e permitindo o reuso parcial dos modelos. Como desvantagem, pode-se citar o fato de que mesmo nos níveis mais altos de abstração, há um uso de informações específicas como tipos primitivos.

13 http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

### 3.2.2 Abordagem de Daniele *et al*

O trabalho de Daniele *et al* tem como base o trabalho e a linguagem apresentada por Almeida *et al*, mas com um enfoque diverso [Daniele *et al*, 2009], buscando aplicar os princípios dessa abordagem para um grupo diferente de aplicações. Conforme mostrado na Figura 3.3, os autores apresentam uma abordagem baseada em *Model-Driven Architecture* (MDA) para a modelagem do comportamento de aplicações móveis e sensíveis ao contexto, utilizando a plataforma A-MUSE [TAKAHASHI, SUGANUMA e SHIRATORI, 2005] como base e as linguagens de Almeida *et al*, para a modelagem, mantendo toda a estrutura de níveis de abstrações, mas modificando o objetivo da linguagem e a forma das transformações (além da linguagem ECA-DL ser renomeada como "A-MUSE DSL" neste trabalho).

O trabalho é focado nos dois primeiros níveis de abstração,*service specification* e *service design refined model* e nas relações de transformações entre eles. O *service specification* é representado de maneira idêntica ao trabalho de Almeida *et al*, com os atributos das informações contextuais da aplicação descritos em UML e os aspectos comportamentais da aplicação descritos em "A-MUSE DSL" (ECA-DL) e é definida como sendo as funcionalidades oferecidas pela aplicação em termos de ações e relações de casualidade entre estas ações. O segundo nível, *service design refined model*, recebe ações e relações, e identifica e modela sequencias de ações, chamadas de *interaction patterns*. Esses padrões são utilizados para a criação dos componentes existentes para as aplicações sensíveis ao contexto. Essa correlação é mostrada na Figura 3.3.

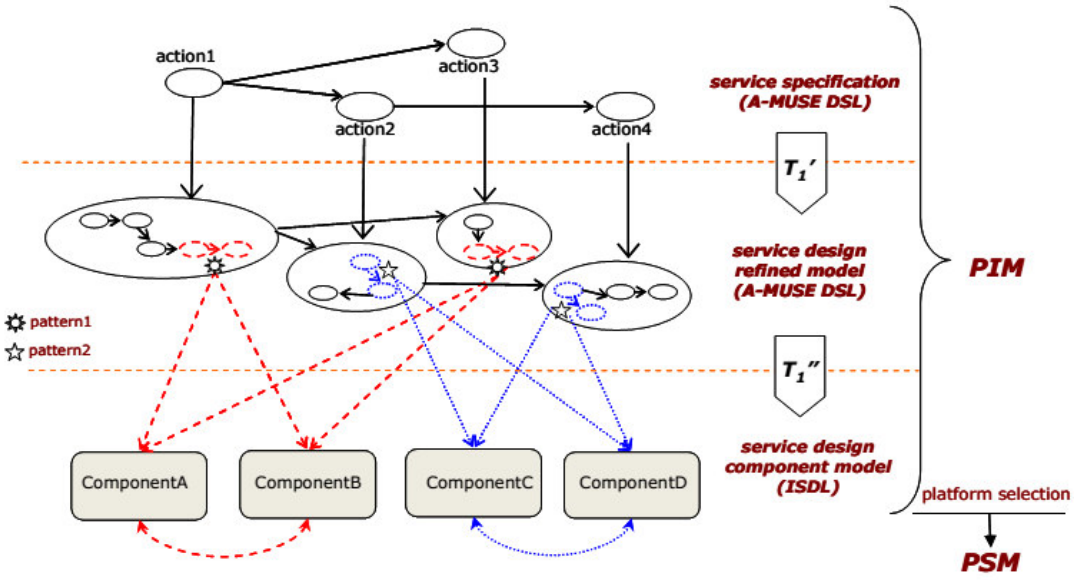


**Figura 3.3 – Os três níveis de modelos da abordagem de Daniele *et al* [2009]**

A Figura 3.4 mostra uma visão geral da primeira transformação, utilizando o Medini QVT. A segunda transformação leva ao terceiro nível, chamado *service design component model*, que descreve o comportamento individual dos componentes das aplicações, modelados em ISDL.

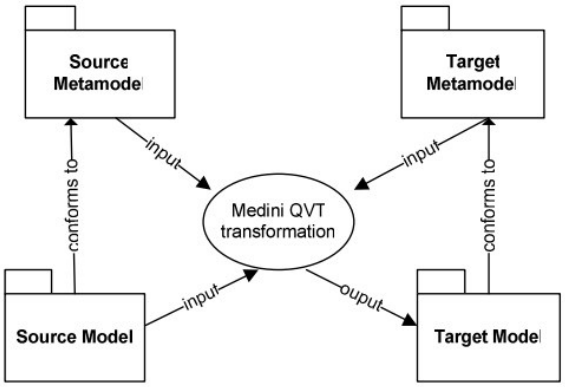


**Figura 3.4 – Visão geral da transformação realizada pelo Medini QVT [Daniele *et al*, 2009]**

Entre as vantagens apresentadas pela abordagem de Daniele *et al*, pode-se mencionar um maior em aplicações móveis e sensíveis ao contexto, com o desenvolvimento de componentes para estas aplicações. No entanto, por utilizarem as mesmas linguagens de Almeida *et al*, esta abordagem apresenta os mesmos problemas, tais como o excesso de informações de nível mais específico em modelos de alto nível.

### 3.1.3 Copal-DSL

O COPAL (*Context Provisioning for ALl*) é um *middleware* que provê baixo acoplamento entre a camada de aquisição de contexto e as aplicações [LI, SEHIC e DUSTDAR, 2010]. A parte principal da arquitetura do COPAL é o *framework* de mesmo nome que é dividido em sete componentes que se relacionam: *ContextType*, *Publisher*, *Action*, *Processor*, *ContextEvent*, *ContextQuery* e *Listener*. A Figura 3.5 exibe os componentes do COPAL e os relacionamentos existentes entre eles.

Baseado nesse relacionamento entre os componentes e em modelos de aquisição de contexto foi desenvolvida a COPAL-DSL, que modela a aquisição das informações contextuais de maneira compatível com este *middleware*.

A COPAL-DSL é uma linguagem textual, escrita em xText e com transformações em xPand, que engloba dois grupos de modelos: *Component Models* e *Deployment Models*. Os *Component Models* modelam e especificam a estrutura e atributos de cada componente do

*framework*. Os *Deployment Models* especificam como os artefatos gerados pelos modelos de primeiro tipo serão transformados em *bundles* OSGi equivalentes.

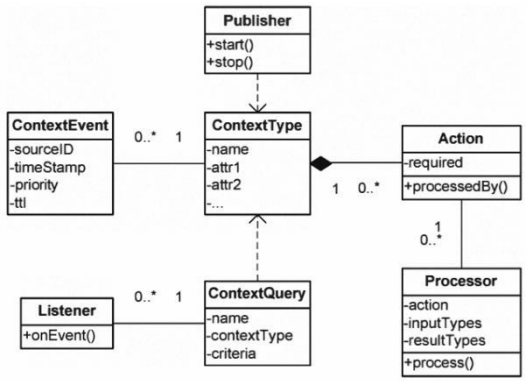


**Figura 3.5 – Componentes do COPAL [LI, SEHIC e DUSTDAR, 2010]**

A Figura 3.6 apresenta os *Component Models* e os seus artefatos. A DSL é composta por cinco classes básicas: *Publisher*, *ContextType*, *Listener*, *ContextQuery* e *Processor*.

A classe *Publisher* define um único contexto, que é especificado pela classe *ContextType*. Essa classe provê atributos e *defaultActions*, ações que podem ser aplicadas por esse contexto.O *Listener* indica quais eventos são de seu interesse através de *queries* especificadas pela *ContextQuery*. Os *listeners* são registrados no COPAL, onde cada *Listener* está associado a uma *ContextQuery*.

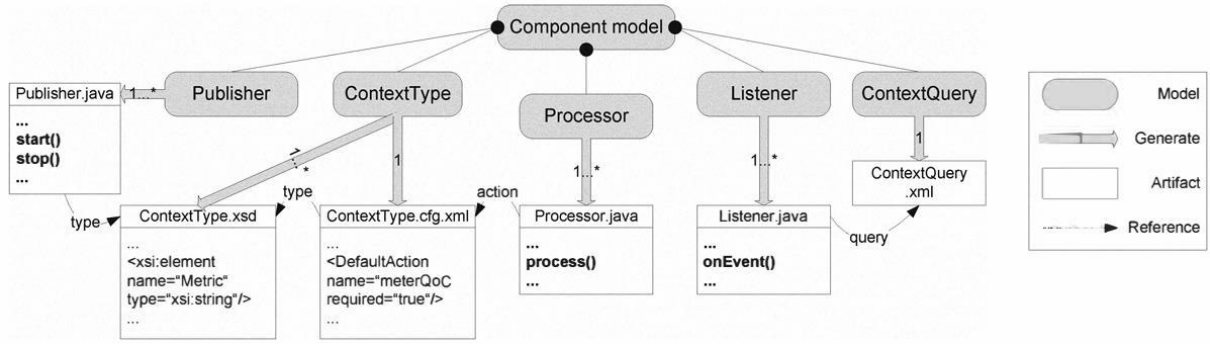


**Figura 3.6 – *Component Models* e Artefatos do COPAL-DSL [LI, SEHIC e DUSTDAR, 2010]**

A classe *ContextQuery* define os critérios através dos quais os valores de uma informação contextual podem ser de interesse da aplicação. Essas informações contextuais são tratadas sempre como eventos pelo COPAL e esses valores podem ser instanciados com operadores aritméticos ou lógicos.

A classe *Processor* é a responsável por indicar uma ação. Essa ação é associada a um evento, que é a condição para ocorrência da ação. Na classe *Processor*, o evento é definido como parâmetro de entrada e ação a ser realizada como parâmetro de saída.

Os *Deployment Models*, apresentados na Figura 3.7, são divididos em *manifest model* e *artifacts model*. O primeiro registra as informações gerais e o nome do *main activator*, que está relacionado com o *artifacts model*. O segundo armazena os nomes dos *publishers*, *listerners* e *processors* que serão implementados [LI, SEHIC e DUSTDAR, 2010].

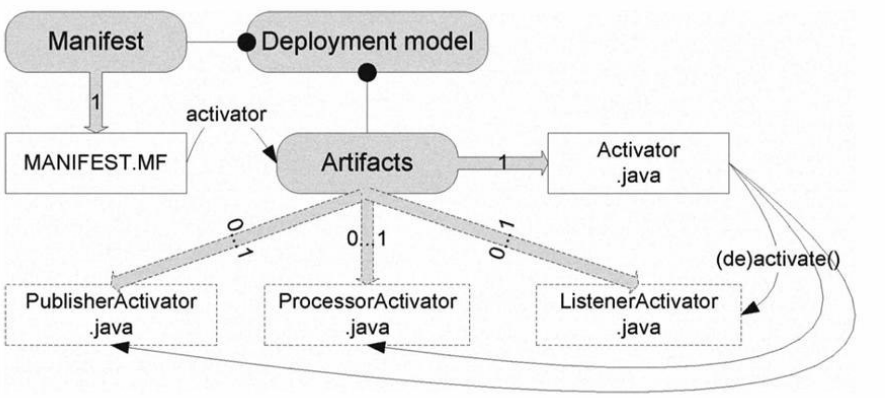


**Figura 3.7 – *Deployment Models* e Artefatos do COPAL-DSL [LI, SEHIC e DUSTDAR, 2010]**

A COPAL-DSL apresenta como vantagens ter elementos de sintaxe de alto nível de abstração, facilitando o seu entendimento e a própria modelagem das informações contextuais. Além disso, a linguagem foi desenvolvida utilizando tecnologias consolidadas, tais como xText e xPand. Isso auxilia no processo de entendimento do processo de modelagem para o desenvolvedor. No entanto, uma grande desvantagem da COPAL-DSL é a sua dependência do *middleware* COPAL, uma vez que ela foi desenvolvida especificamente para a modelagem de informações contextuais para o COPAL. De fato, os próprios autores frisam que, para o desenvolvimento de aplicações utilizando a COPAL-DSL, faz-se necessário o entendimento dos componentes do COPAL e dos relacionamentos entre esses [LI, SEHIC e DUSTDAR, 2010].

## 3.1.4 MLContext

A MLContext é uma DSL textual desenvolvida para a modelagem de informação contextual e geração automática de artefatos de softwares a partir dos modelos construídos [HOYOS, GARCÍA-MOLINA e BOTÍA, 2010]. Ela trabalha com a abordagem de separar a definição dos elementos contextuais dos detalhes envolvendo as fontes destas informações contextuais ou aspectos do contexto que sejam dependentes das aplicações [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013].

O modelo de contexto é formado por uma entidade, onde cada contexto é uma entidade. O contexto pode ser simples ou composto (uma agregação de vários contextos

simples). Cada contexto simples é formado por uma ou mais informações contextuais de um mesmo tipo (*physical*, *environment*, *computational* (*System*), *personal*, *social* e *task*). O valor de uma informação contextual pode ser constante ou pode ser oriunda de uma fonte de contexto. A Figura 3.8 apresenta o metamodelo da MLContext.

O metamodelo do MLContext foi implementado na linguagem TCS[14], da plataforma Eclipse. A Figura 3.9 mostra um exemplo de como são os modelos na linguagem MLContext. Modelos produzidos em MLContext não contém informações específicas de plataforma ou aplicação. Assim, a MLContext permite o reuso de um mesmo modelo contextual em diversas aplicações e para diferentes plataformas de *middleware*, desde que existam as regras de transformação especificas para a plataforma-destino. Existem transformações que geram artefatos compatíveis com duas plataformas de *middleware*: OCP [BOTIA *et al*, 2009] e JCAP [BARDRAM, 2005], atualmente [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013].

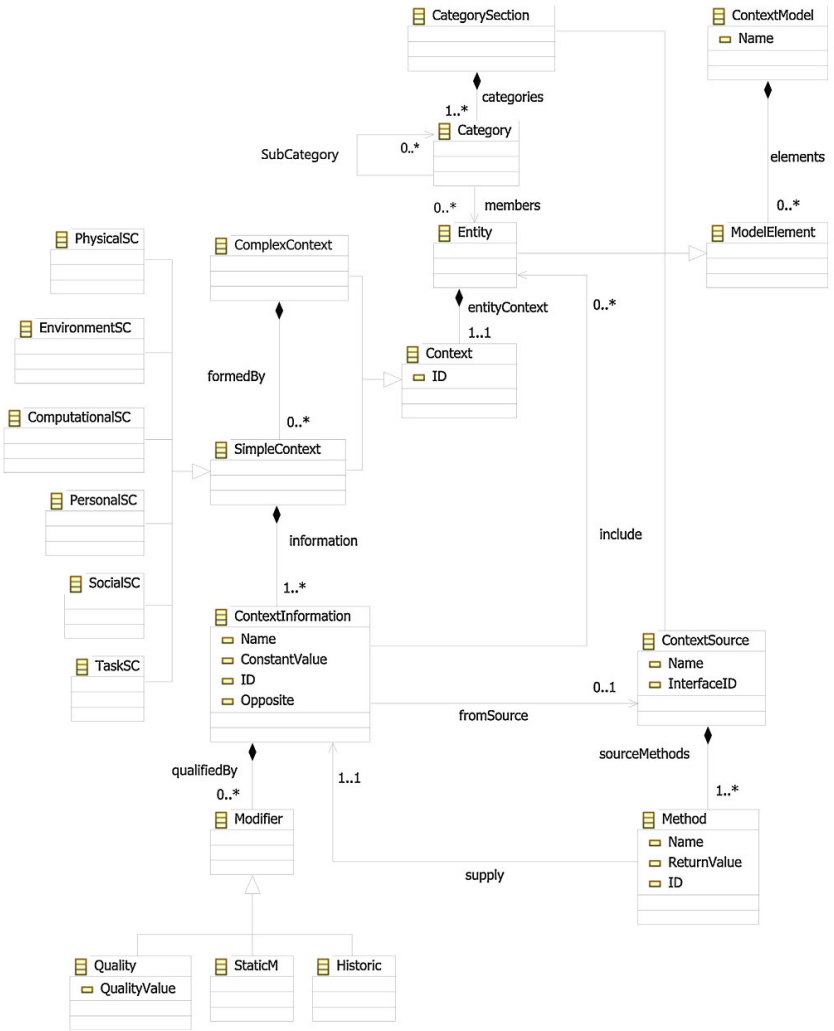


**Figura 3.8 – Metamodelo do MLContext [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]**

14 http://www.eclipse.org/gmt/tcs/

```
entity floor_01 context {
   environment { "contains" : room_01, room_02 static
   physical    { "fire_presence" source } }

 entity room_01 context {
    environment { "contains" : bed_01 , tv_01 static
    physical    { "temperature" source } }

        entity tv_01 context {
           computational { "status" source } }

        entity bed_01 context {
           environment { "contains" : person_02 } }

        entity person_02 context {
           personal { "name" :      "John"  static
                      "surname" :   "Smith" static
                      "age" :       "39" }
           social {   "role" :      "patient" }
           physical { "temperature" source  historic
                      "pulse"       source } }
```

**Figura 3.9 – Exemplo de um trecho de modelo do MLContext [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]**

As transformações, específicas para cada plataforma, foram implementadas com MOFScript e geram representações contextuais diferentes. Esses artefatos gerados dependem de como cada plataforma de *middleware* trabalha com as informações contextuais. No caso do OCP, o artefato final da modelagem será uma ontologia OWL-DL (*Web Ontology Language*[15]), além de um esqueleto do código Java para a inicialização das instâncias contextuais. No caso do JCAP, uma série de classes Java equivalentes ao modelo é gerada. A Figura 3.10 apresenta a estratégia de geração de código adotada pelo MLContext para manter-se independente de plataforma de *middleware*. Toda a lógica da arquitetura interna das plataformas de *middleware* foi encapsulada nas APIs de transformação, deixando o modelo construído ainda em alto nível.

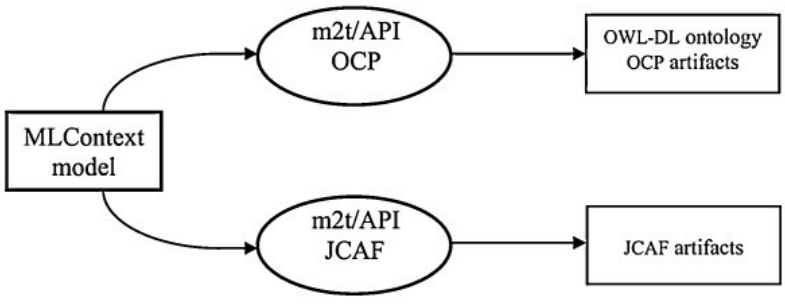


**Figura 3.10 – Estratégia de geração de código independente de plataforma de *middleware* da MLContext [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]**

15 http://www.w3.org/TR/owl-features/

A abordagem do MLContext tem uma importante característica: a independência de plataforma de *middleware*. De fato, a estratégia utilizada por essa abordagem em encapsular a lógica da arquitetura internas nas APIs de transformação inspirou o uso de uma estratégia similar a da solução proposta nesta dissertação. Além disso, como metamodelo da MLContext tão genérico quanto possível, ele permite o reuso dos modelos construídos para diversas aplicações e plataformas. No entanto, uma lacuna percebida nesta abordagem é a ausência da representação de regras contextuais ou mesmo de um suporte à adaptação em tempo de execução no caso da ocorrência de determinados eventos ou da satisfação de certas condições.

### 3.1.5 Abordagem de Santos *et al*

O trabalho de Santos *et al* se propõe a desenvolver uma DSL independente de plataforma para a especificação de processos de inferências de contexto adaptáveis dinamicamicamente e uma plataforma de *middleware* que sirva de suporte infraestrutural para a interpretação e execução dos processos de inferência executados por essa DSL [SANTOS *et al*, 2011]. A Figura 3.11 apresenta a arquitetura do *middleware*, bem como a correlação entre *middleware* e DSL. Um dos componentes internos do *middleware* é o interpretador da linguagem. Dessa forma, recebe os modelos definidos pela DSL, realiza as transformações desses modelos internamente e utiliza os artefatos gerados para gerenciar a inferência do contexto. Com base nesses artefatos interpretados e nas informações contextuais recebidas pelos sensores, o *middleware* realiza a inferência do contexto e o repassa para a aplicação.

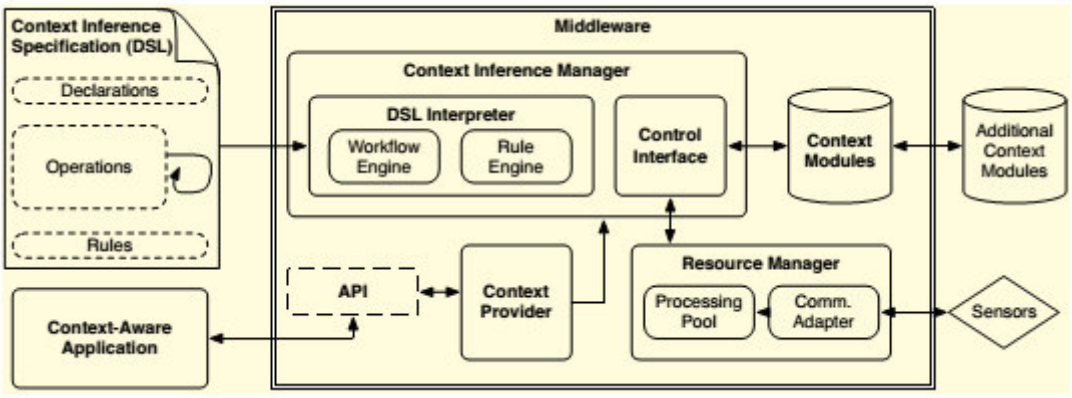


**Figura 3.11 – Arquitetura do *middleware* e da DSL [SANTOS *et al*, 2011]**

Ao juntar as duas abordagens, uma plataforma de *middleware* e uma DSL, Santos *et al* objetiva facilitar o desenvolvimento de aplicações móveis que precisem utilizar informações contextuais e se adaptar a elas de acordo com as inferências obtidas. Assim, é obtida uma maior transparência no uso das informações contextuais, uma vez que todo o processo de

aquisição e inferência é de responsabilidade do *middleware*. A Figura 3.12 apresenta o *workflow* do processo de inferência contextual realizado pela plataforma de *middleware*.

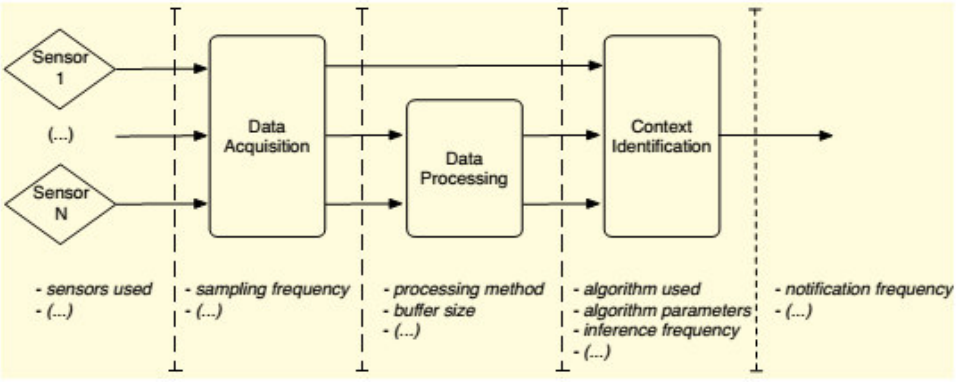


**Figura 3.12 – *Workflow* do Processo de Inferência Contextual [SANTOS *et al*, 2011]**

Tanto a plataforma de *middleware* e quanto a DSL trabalham de maneira biunívoca, uma vez que o próprio interpretador da DSL faz parte da arquitetura do *middleware*, permitindo a interpretação, execução e implementação desta em tempo de execução.

A DSL proposta possibilita o uso de comportamento adaptativo no processo de inferência de contexto, através de um conjunto de regras de adaptação, de acordo com eventos ou condições, e operações de inferência. Um exemplo de regra contextual na linguagem proposta por Santos *et al* está na Figura 3.13. Nesse exemplo, é especificado qual sensor deve ser utilizado para indicar a posição do dispositivo com uma regra que indica mudanças no sensor utilizado de acordado com o nível da bateria do dispositivo e dos sensores disponíveis. O sensor padrão é o GPS, mas a cada cinco minutos é realizada uma verificação do nível de bateria e dos sensores disponíveis. Dependendo das condições observadas, o sensor utilizado pode ser mudado para o WiFi ou o GSM.

```
 1: infPos IS positionInf(sensor=SENSOR.GPS){position};
 2:
 3: RUN[period=5sec]{
 4:    context = infPos();
 5: }
 6:
 7: RULES{
 8:    EVERY 5min: ENERGY.LEVEL < 50%
 9:                && SENSOR.GSM.AVAILABLE{
10:       infPos IS positionInf(sensor=SENSOR.GSM);
11:    }
12:    EVENT: SENSOR.GPS.UNAVAILABLE
13:           && SENSOR.WiFi.AVAILABLE{
14:       infPos IS positionInf(sensor=SENSOR.WiFi);
15:    }
16:    EVENT: SENSOR.WiFi.UNAVAILABLE
17:           && SENSOR.GSM.AVAILABLE{
18:       infPos IS positionInf(sensor=SENSOR.GSM);
19:    }
20: }
```

**Figura 3.13 – Exemplo de Regra na DSL de Santos *et al* [2011]**

A DSL é dividida, primordialmente, em três partes principais: *declarations*, *operations* e *rules*.

*Declarations* é a parte estrutural inicial da DSL, que declara os métodos, variáveis e valores de parâmetros que serão usados pelas *operations*, ligando os elementos da DSL com os componentes do *middleware*.

*Operations* é a parte que define e especifica o processo de inferência, que este pode ser dividido em três etapas: *data acquisition*, através dos sensores do dispositivo, *data processing* e, por fim, *context identification*, responsável por retornar as informações contextuais obtidas.

*Rules* é a parte que especifica as estratégias de adaptação que serão aplicadas em caso de ocorrência das condições ou eventos previamente determinadas. As *rules* são responsáveis por promover a adaptação, ajustando o comportamento do processo de inferência da sua respectiva *operation*. Em caso de múltiplas *rules* por *operation*, existe uma priorização especificada de modo a evitar conflitos [SANTOS *et al*, 2011].

A *posteriori*, em [SANTOS *et al*, 2013], foi definida um quarto componente da linguagem chamado de *code*. *Code* seria a parte a permitir que o desenvolvedor adicionar funcionalidades que não estivessem presentes na DSL. Na Figura 3.14, é mostrada uma visão geral da política de adaptação do contexto, de acordo com as *Rules* e as *Operations*. Esta política de adaptação de regras contextuais serviu de inspiração para o desenvolvimento da parte da modelagem de regras contextuais no metamodelo utilizado na solução proposta nessa dissertação.

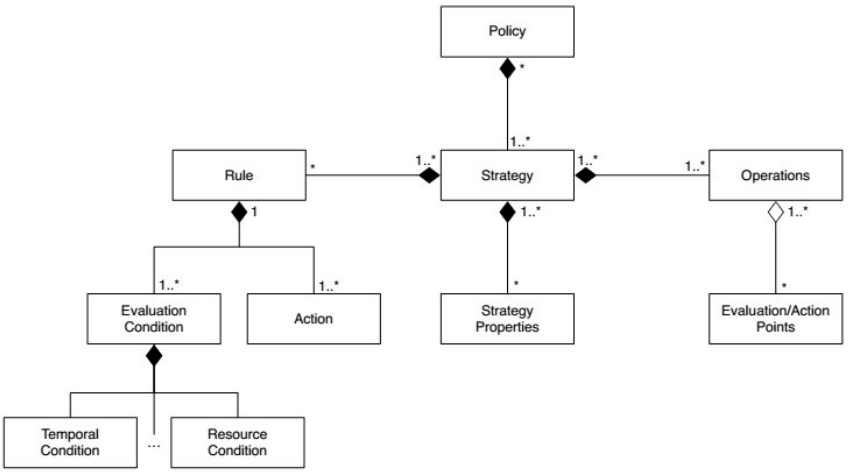


**Figura 3.14 – Visão Geral do Modelo para a Política de Adaptação [SANTOS *et al*, 2013]**

A abordagem de Santos *et al* apresenta diversas vantagens, dentre as quais se podem mencionar o alto grau de transparência, o suporte a adaptação em tempo de execução e a

representação de regras apresentada de uma forma de fácil entendimento. No entanto, essa abordagem também apresenta grandes desvantagens. A principal delas é o alto grau de acoplamento entre linguagem e *middleware*, de tal forma que um precisa necessariamente do outro para poder ser utilizado. Como o próprio *middleware* da abordagem realiza a interpretação dos modelos construídos pela DSL, a inexistência desses artefatos impossibilita o seu uso.

## 3.2 Características de DSLs para Computação Móvel e Sensível ao Contexto

Durante a revisão de literatura, foi levantado um conjunto de características que estariam entre as mais apropriadas para o domínio de DSL desta dissertação. Assim, foram buscadas DSLs voltadas para o desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizem plataformas de *middleware*. Essas características proveem uma linha-guia para o desenvolvimento de novas DLS para aplicações móveis e sensíveis ao contexto.

### 3.2.1 Metodologia

Para se realizar a elicitação das características, a primeira etapa foi a leitura dos trabalhos e abordagens apresentados na Subseção anterior. Com base nas comunalidades e nas particulares de cada DSL, foi reunido um conjunto de características relevantes apresentadas por elas. Foram analisados se essas características entravam em conflito umas com as outras ou se algumas dessas características não passariam de um caso específico de outras propriedades. Após descartar algumas delas, foram analisadas se as características encontradas eram realmente características significativas ou apenas características desejáveis de uma DSL.

Hoyos *et al* [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013] apresentam um conjunto de seis requisitos (Abstrações de alto-nível; Tipos de contexto; Modelos independentes de plataforma; Reuso de Modelos e independência do domínio da aplicação; Rastreabilidade, qualidade e mecanismos temporais; Facilidade de uso) que norteiam o desenvolvimento da DSL apresentada no trabalho, a MLContext. Esses requisitos estão de acordo com a experiência obtida com a utilização da plataforma de *middleware* OCP. Esses requisitos apresentados por Hoyos *et al* também foram analisados em termos de significância e possíveis conflitos com outros. Alguns deles foram inseridos no conjunto de características apresentadas na Subseção a seguir, enquanto outros foram descartados. As razões para o

descarte deles deveu-se ao fato de serem subjetivos demais ou não servirem como critério para comparação ou diferenciação entre as DSLs existentes.

### 3.2.2 Características elencadas

As características levantadas para a construção de uma DSL para o domínio de aplicações móveis e sensíveis ao contexto que utilizem plataformas de *middleware* para o gerenciamentos das informações contextuais são as seguintes:

- **Independência de Plataforma de *Middleware*:** A DSL deve ser independente da plataforma de *middleware*. Uma das vantagens da utilização dos conceitos de MDE é a possibilidade que, com um mesmo modelo, haja a capacidade de gerar aplicações para diferentes plataformas. Assim, no caso das DSLs da natureza estudada neste trabalho, uma característica importante seria que a linguagem não fosse especifica para uma determinada plataforma de *middleware*. Uma DSL dependente de plataforma de *middleware* não é necessariamente "problemática". No entanto, uma DSL independente de plataforma pode gerar modelos que podem ser utilizados por diversas plataformas de *middleware* ou por diversos Sistemas Operacionais móveis. Além disso, utilizando uma DSL independente de plataforma, os desenvolvedores não precisariam conhecer os aspectos da arquitetura interna da plataforma de *middleware*.
- **Tipos de Contexto:** Aplicações móveis e sensíveis ao contexto complexas lidam com um grande conjunto de elementos contextuais, que são freqüentemente categorizadas em tipos de contexto (e.g., *physical, environment, social, etc*.) [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]. Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve oferecer modelos que provenham uma diversidade de tipos de contexto. Segundo Hoyos *et al* [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013], uma DSL que contenha esse tipo de recurso oferece um esquema de armazenamento dos modelos mais eficiente e próximo do mundo real. Assim, com um conjunto de tipos de contextos, os desenvolvedores são capazes de caracterizar melhor as situações de contexto com que a plataforma de *middleware* ou as aplicações devem lidar.
- **Reuso de Modelos:** Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve fornecer modelos capazes de serem reutilizados para outras plataformas de *middleware* e aplicações sensíveis ao contexto. As informações contextuais têm pontos em comum, mesmo quando são utilizadas por diferentes aplicações. A reutilização é uma das principais vantagens da abordagem MDE

[HOYOS, GARCÍA-MOLINA e BOTÍA, 2013] e as DSLs devem maximizar essa vantagem.

- **Transparência:** Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve prover transparência ao desenvolvedor quando este lida com uma informação contextual. Os desenvolvedores não devem ter conhecimento sobre como os processos de aquisição e de inferência contextuais são realizados. O gerenciamento de contexto deve ser provido pelas plataformas de *middleware*. Na verdade, sua abstração é uma característica transversal no desenvolvimento de uma plataforma de *middleware* pervasiva e sensível ao contexto [MAIA, ROCHA e ANDRADE, 2009].
- **Suporte à Adaptação:** É uma característica transversal para aplicações móveis e sensíveis ao contexto e modelar esse comportamento é algo esperado para uma DSL voltada para o desenvolvimento de aplicações móveis e sensíveis ao contexto [MAIA, ROCHA e ANDRADE, 2009].
- **Representação de Regras:** Uma DSL voltada para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve fornecer uma representação de regras, permitindo a criação de regras condicionais para a aquisição da informação contextual e, em alguns casos, permitir a especificação do comportamento adaptativo.
- **Ferramenta de Transformação:** Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve usar uma ferramenta de transformação que não seja fortemente acoplada nem à DSL e nem à plataforma de *middleware*. Com o objetivo de permitir a independência de plataforma, a ferramenta deve fornecer regras de transformação para diferentes plataformas de *middleware*. Assim, a DSL deve usar uma ferramenta de transformação que satisfaça essa característica.
- **Nível Independente de Plataforma:** Uma DSL pode ter vários níveis de abstração. Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve ter, pelo menos, um nível de abstração que é independente de plataforma. O número ideal de níveis de abstração é uma questão subjetiva e depende de como a DSL foi projetada. A DSL pode seguir o padrão MDA, fazendo uma transformação direta de código ou através de outros níveis de abstração.
- **Geração de Artefatos:** Uma DSL para o desenvolvimento de aplicações móveis e sensíveis ao contexto deve gerar como produto final das suas transformações um artefato de desenvolvimento. Esse artefato deve reduzir a complexidade para o

desenvolvimento de aplicações sensíveis ao contexto.

### 3.1.3 Priorização das Características

As características apresentadas na Subseção anterior não têm a prioridade e relevância equivalente. Eles podem ser divididos em dois grupos. Isso se deve ao fato de que um grupo de características apresenta funcionalidades mais essenciais que o outro grupo.

No primeiro grupo, pode-se citar: Independência de Plataforma de *Middleware* e Suporte à Adaptação. Essas duas características devem ser importantes para qualquer DSL no domínio de aplicações móveis e sensíveis ao contexto. No entanto, ambas as características são satisfeitas por poucas abordagens.

No segundo grupo, pode-se citar: Tipos de Contexto, Reuso de Modelos, Transparência, Ferramenta de Transformação, Representação de Regras, Nível Independente de Plataforma e Geração de Artefatos. Essas características estão presentes na maioria das abordagens estudadas, ainda que parcialmente. No entanto, cada abordagem utiliza estratégias e técnicas distintas para prover tais características. Algumas dessas características sobrepõem-se a outras, de modo que podem ser vistos como refinamentos desses sobrepostos. Um exemplo é o Suporte à Adaptação e a Representação de Regras. Outro exemplo é a Independência de Plataforma de *Middleware*, que para poder ser realizada necessita que outras características do segundo grupo estejam presentes (e.g., Reuso de Modelos, Ferramenta de Transformação, Nível Independente de Plataforma).

### 3.2.3 Comparativo DSLs x Características

A Tabela 3.1 apresenta um comparativo entre as abordagens apresentadas na Seção 3.1 no tocante as características descritas na Seção 3.2.

**Tabela 3.1 – Comparação dos Trabalhos Relacionados de acordo com as Características elencadas**

| Caracterís-ticas | DSLs | | | | |
|---|---|---|---|---|---|
| | **Abordagem de Almeida *et al*** | **Abordagem de Daniele *et al*** | **Copal-DSL** | **MLContext** | **Abordagem de Santos *et al*** |
| Independência de Plataforma de *Middleware* | Não (A-MUSE) | Não (A-MUSE) | Não (COPAL) | Sim | Não (plataforma de *middleware* proprietária) |
| Tipos de Contexto | Sim | Sim | Sim | Sim | Não |
| Reuso de Modelos | Parcial | Parcial | Não | Sim | Não |
| Transparência | Sim | Sim | Sim | Sim | Sim |
| Suporte à Adaptação | Não | Não | Não | Não | Sim |
| Representação de Regras | Parcial | Parcial | Não | Não | Sim |
| Ferramenta de Transformação | GReAT, uma ferramenta baseada em GME | Medini QVT | XPand | MOFScript | - |
| Nível Independente de Plataforma | Sim | Sim | Não | Sim | Não |
| Geração de Artefatos | Uma especificação de um serviço de contexto dependente de plataforma | Uma especificação de uma aplicação móvel e sensível a contexto | Bundles OSGi | Esqueleto de código Java para inicialização e instâncias contextuais | - |

#### 3.2.6.1 Independência de Plataforma de *Middleware*

A maior parte das DSLs estudadas na Subseção 3.2 são dependentes de uma plataforma de *middleware* específica. Um exemplo é a COPAL-DSL, que foi desenvolvida focada nos componentes da arquitetura do *framework* COPAL [LI, SEHIC e DUSTDAR, 2010]. No que se refere à independência de plataforma, apenas a MLContext é a única abordagem proposta que conseguiu satisfazer essa característica. Com a sua DSL, é possível modelar o contexto de forma genérica [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013].

Todas as demais DSLs das outras abordagens são dependentes da plataforma *middleware* e não proveem a possibilidade de reuso completo do modelo gerado. No entanto,

Almeida *et al* [ALMEIDA *et al*, 2006] e Daniele *et al* [DANIELE *et al*, 2009] se aproximam disso, uma vez que suas abordagens permitem o reuso parcial, já que o modelo do primeiro nível de abstração é independente de plataforma. No entanto, a partir do modelo de segundo nível de abstração, os modelos se tornam dependentes do *middleware* A-MUSE [TAKAHASHI, SUGANUMA e SHIRATORI 2005] para especificação do contexto de serviço.

#### 3.2.6.2 Tipos de Contexto

A maior parte das DSLs apresentadas fornece suporte a tipos de contexto. Em MLContext, por exemplo, o modelo de contexto consiste de uma lista de entidades. Um contexto pode ser um elemento de contexto simples (por exemplo, a localização do usuário) ou uma agregação de vários contextos simples. Cada contexto simples é formado por um ou mais informações contextuais do mesmo tipo (e.g., *physical*, *environment*, *computational*, *personal*) [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013].

Em COPAL-DSL, os tipos de contexto são especificados pela classe *ContextType*. Cada tipo de contexto (por exemplo, a bateria) tem um *ContextType* equivalente. Nesta classe, o contexto é definido por um conjunto de atributos (por exemplo, o valor, métricas, duração) e *defaultActions*, ou seja, ações que podem ser aplicadas neste tipo de contexto[LI, SEHIC e DUSTDAR, 2010].

#### 3.2.6.3 Reuso de Modelos

Em MLContext, o mesmo modelo contextual pode ser reutilizado para a modelagem em diversas aplicações e plataformas de *middleware* [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013], reduzindo o tempo de desenvolvimento e custos.

Nas abordagens de Almeida *et al* [ALMEIDA *et al*, 2006] e Daniele *et al* [DANIELE *et al*, 2009], os primeiros níveis de abstração também podem ser reutilizados em várias aplicações desenvolvidas nessas linguagens.

#### 3.2.6.4 Transparência

Todas as abordagens satisfazem a característica de transparência. A abordagem que oferece o maior grau de transparência ao desenvolvedor é a abordagem de Santos *et al* [SANTOS *et al*, 2011].

Neste trabalho, após a modelagem de contexto ser realizada, o desenvolvedor precisa apenas realizar uma requisição da informação contextual inferida pelo middleware.

#### 3.2.6.5 Suporte à Adaptação

A DSL apresentada por Santos *et al* [SANTOS et al, 2011] permite a modelagem do comportamento adaptativo da aplicação. A sua DSL oferece uma linguagem para modelagem de regras de adaptação, que são ativados por eventos, condições e operações de inferência.

A COPAL-DSL oferece outra forma de adaptação. Esta DSL define ações (*actions*), critério (*criteria*) e *listeners*. Quando um *listener* informa que o valor de um atributo corresponde a um critério especificado, uma ação é executada [LI, SEHIC e DUSTDAR, 2010].

#### 3.2.6.6 Representação de Regras

Nos primeiros níveis de abstração das abordagens Almeida *et al* [ALMEIDA *et al*, 2006] e Daniele *et al* [DANIELE *et al*, 2009], a ECA-DL (*Events-Conditions-Actions Domain Language*) é utilizada. A ECA-DL é uma especialização de uma linguagem de propósito geral, a IDSL. Com esta linguagem, um engenheiro de software pode especificar o comportamento em termos de ações e as relações de causalidade entre essas ações.

A DSL de Santos *et al* [SANTOS *et al*, 2011] é dividida em quatro partes principais e uma delas é se chama *Rules*. *Rules* é a parte que especifica as estratégias de adaptação a serem implementadas em caso de ocorrência de condições ou eventos pré-determinados. As *Rules* são responsáveis por promover a adaptação, ajustando o comportamento do processo de inferência. Em caso de múltiplas *rules* por operação, há uma ordem de prioridades especificadas para evitar conflitos [SANTOS *et al*, 2011].

#### 3.2.6.7 Ferramenta de Transformação

Quase todas as abordagens apresentadas utilizam ferramentas ou linguagens de transformação que são consolidadas na comunidade e que estejam disponíveis. Por exemplo, as duas transformações que ocorrem na DSL descrita por Almeida *et al* [ALMEIDA *et al*, 2006] utilizam a ferramenta GReAT (*Graph Rewriting And Transformation*). Por sua vez, as transformações no Daniele *et al* [DANIELE *et al*, 2009] são implementadas na ferramenta Medini QVT.

As transformações que são realizadas na MLContext são executadas com MOFScript [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013] e as transformações realizadas na COPAL-DSL são executadas com xPand [LI, SEHIC e DUSTDAR, 2010]. Apenas a abordagem de Santos *et al* não utiliza uma ferramenta de transformação consolidada ou disponível. Sua estratégia consistiu em embarcar o interpretador da DSL dentro do *middleware*.

#### 3.2.6.8 Nível Independente de Plataforma

Almeida *et al* [ALMEIDA *et al*, 2006] propõe uma DSL para a construção de serviços sensíveis ao contexto, divididos em três níveis de abstração, seguindo o padrão da MDA. Por outro lado, a MLContext tem apenas um nível de abstração.

No entanto, suas transformações são específicas para cada plataforma, gerando uma representação contextual própria para cada *middleware*. Assim, a MLContext tem um nível de abstração independente de middleware [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013].

#### 3.2.6.9 Geração de Artefatos

A DSL de Almeida *et al* [ALMEIDA *et al*, 2006], por exemplo, gera uma especificação para uma plataforma de serviço particular, como BPEL ou *Web Service*. A abordagem de Daniele *et al* [DANIELE *et al*, 2009] gera sequências de ações, chamadas de *interaction patterns*. Esses *interaction patterns* são utilizados para criar os componentes existentes para aplicações móveis e sensíveis ao contexto.

Em COPAL-DSL, os modelos são transformados em *bundles* OSGi equivalentes [LI, SEHIC e DUSTDAR, 2010]. Em MLContext, o artefato final varia de acordo com a plataforma de *middleware*. No caso da OCP, o artefato final é uma ontologia OWL-DL, e um esqueleto de código Java. No caso de JCAP, o resultado é um conjunto de classes Java [HOYOS, GARCÍA-MOLINA e BOTÍA, 2013]. Apenas a abordagem de Santos *et al* não produz um artefato de software como produto final das transformações realizadas.

## 3.4 Conclusão

Neste capítulo, foram apresentados os trabalhos relacionados a esta pesquisa e um conjunto de características desejadas em uma DSL para o domínio de estudo dessa dissertação.

As características foram elencadas durante a revisão de literatura, analisando as comunalidades e variabilidades dos trabalhos encontrados. Chegou-se a um grupo de nove características importantes: Independência de Plataforma de *Middleware*, Tipos de Contexto, Reuso de Modelos, Transparência, Suporte à Adaptação, Representação de Regras, Ferramenta de Transformação, Nível Independente de Plataforma e Geração de Artefatos.

Durante a revisão de literatura, foram encontradas cinco abordagens diferentes que apresentam DSLs voltadas ao domínio de aplicações móveis e sensíveis ao contexto que utilizem plataformas de *middleware* como forma de aquisição das informações contextuais.

Foram descritas tanto o *middleware* quanto as DSLs utilizadas pelas abordagens, analisando os tipos de transformação realizados por elas, os artefatos de software produzidos, o tipo de DSL projetada e, quando possível, o seu metamodelo. Através da análise das suas comunalidades, das vantagens e das desvantagens apresentadas pelas abordagens, buscou-se encontrar um conjunto de características que fossem importantes no desenvolvimento de abordagens dessa natureza.

Quando realizada a comparação entre as características e quais as abordagens que os atendiam, chegou-se à conclusão de que a MLContext se destaca positivamente nos quesitos independência de plataforma e reuso de modelos, enquanto a abordagem de Santos *et al* é mais vantajosa quando se trata do suporte à adaptação e representação de regras. No entanto, nenhuma das abordagens consegue prover todas as características elencadas de forma satisfatória.

Desta forma, é possível observar que há uma lacuna justamente na questão de não haver abordagens na literatura que consigam satisfazer plenamente as características apresentadas para a construção de DSLs desse domínio, ou mesmo de cumprir a maior parte das características propostas.

# Capítulo 4

# CRITiCAL

Este capítulo apresenta a CRITiCAL (*ConfiguRatIon Tool for Context Aware and mobiLe applications*). A Seção 4.1 descreve uma visão geral da ferramenta, bem como dos processos de geração de código e de configuração de *middleware*. A Seção 4.2 apresenta a DSL ContextRuleML, detalhando o seu metamodelo, sua sintaxe e comparando-o com as características levantadas na Seção 3.2. A Seção 4.3 apresenta a evolução que o *middleware* LoCCAM teve durante o desenvolvimento da CRITiCAL e o seu estado atual de desenvolvimento. A Seção 4.4 apresenta a ferramenta de modelagem visual que responde pela interface da CRITiCAL com o usuário. A Seção 4.5 apresenta, em detalhes, o processo de geração de código, assim como detalha os *templates* usados para a transformação e o produto final dessa transformação. A Seção 4.6 apresenta, em detalhes, o processo de instalação e configuração do *middleware* LoCCAM no dispositivo móvel a partir de um modelo gerado pelo CRITiCAL.

## 4.1 Visão Geral

A CRITiCAL é uma ferramenta de modelagem de aplicações móveis e sensíveis ao contexto para a plataforma Android. Ela baseia a sua modelagem no metamodelo da ContextRuleML e o código Java gerado é voltado para aplicações que utilizem o *middleware* LoCCAM para a realização da aquisição das informações contextuais.

A Figura 4.1 apresenta uma Visão Geral da CRITiCAL e do processo de desenvolvimento da aplicação sensível ao contexto. Através da Visão Geral, é possível distinguir quais etapas do processo que são realizadas pela CRITiCAL e quais são externas a ela. Além disso, é possível distinguir os diferentes processos de transformação que ela realiza e como esta se correlaciona com a DSL ContextRuleML e o *middleware* LoCCAM.

Com base no metamodelo da ContextRuleML, foi desenvolvida uma Ferramenta Visual para a modelagem do contexto da aplicação (passo 1). Uma vez que o modelo do contexto tenha sido construído pelo desenvolvedor (passo 2), ele tem duas opções de transformação.

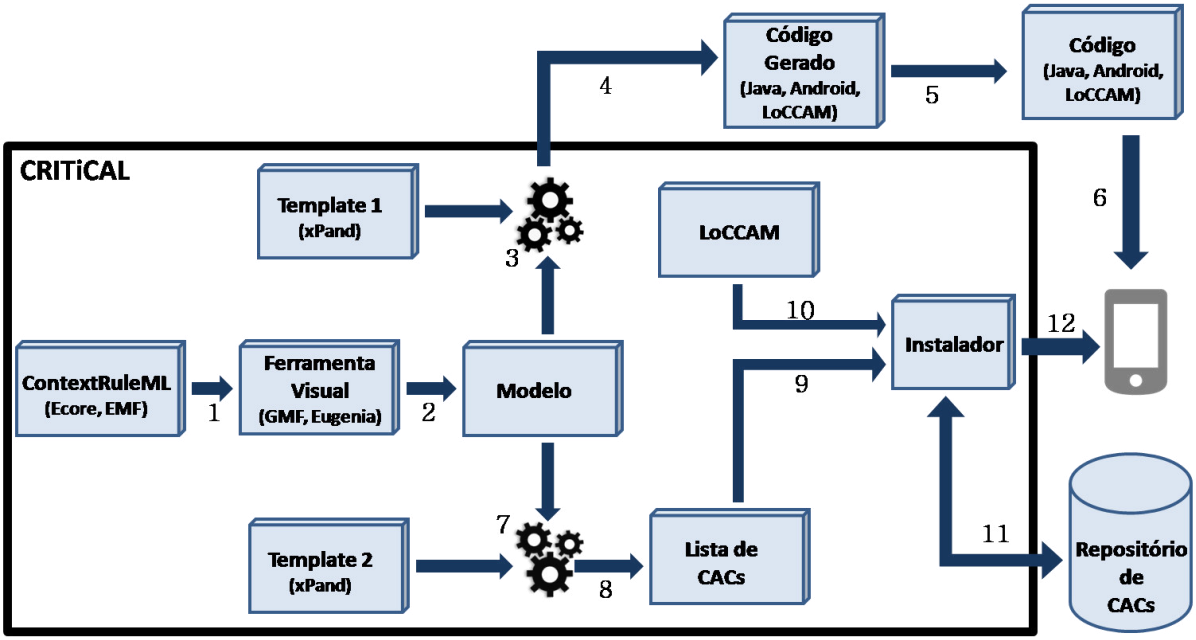


**Figura 4.1 – Visão Geral do Processo**

A primeira opção de transformação (passo 3) realiza a geração do código Java. Para tanto, é usado o modelo construído e um *template* escrito em XPand, resultando em um projeto Android para a IDE Eclipse contendo todos os métodos de comunicação com o LoCCAM e as informações contextuais já gerados (passo 4). Este código inclui a informação contextual monitorada pela aplicação sensível ao contexto (e.g., temperatura, umidade relativa, luminosidade) e as regras contextuais (e.g., notificar a aplicação quando a luminosidade ou a temperatura estiverem em um valor maior que X). Esse projeto Android seria o embrião da aplicação sensível ao contexto a ser desenvolvida (passo 5). Quando esta estivesse completa (passo6), já estaria pronta para instalada em um dispositivo móvel.

A segunda opção de transformação (passo 7) visa a configuração do *middleware* no dispositivo alvo. Através do modelo criado e de outro *template* escrito em XPand, ela gera uma lista de componentes de aquisição de contexto (passo 8) que devem ser instaladas junto com o LoCCAM no dispositivo móvel (e.g., um componente para acessar o sensor de GPS, um componente para capturar os dados do sensor de luminosidade). Utilizando esta lista de CACs, um Instalador seleciona os CACs correspondentes e realiza a instalação dos mesmos (passo 9) e, se preciso, do LoCCAM no dispositivo móvel (passo 10). Esses CACs podem ser selecionados localmente ou baixados automaticamente de um repositório online durante o processo de instalação (passo 12).

### 4.1.1 Princípios de *Design*

A CRITiCAL foi criada com base em *frameworks* consolidados da IDE Eclipse[16]. Isto foi planejado com objetivo de simplificar o uso da ferramenta pelo desenvolvedor, diminuindo a curva de aprendizado e objetivando que todo o processo de desenvolvimento da aplicação fosse restringido a uma única ferramenta.

O metamodelo da ContextRuleML foi construído com base no meta-metamodelo Ecore, através do EMF (Eclipse Modeling Framework)[17]. A ferramenta de modelagem visual foi gerada utilizando o *framework* GMF (Graphical Modeling Framework), com o auxílio da ferramenta Eugenia[18]. Por sua vez, os *templates* das regras de transformação e geração de código foram implementados utilizando xPand, uma linguagem de transformação baseada em EMF. Com o objetivo de simplificar a sintaxe do código gerado, os *templates* foram implementados com base no uso da biblioteca LoCCAM_Lib. Assim, o código gerado torna-se mais entendível pelo desenvolvedor, caso ele, por ventura, decida analisá-lo.

## 4.2 ContextRuleML

Para realizar a modelagem das informações contextuais e das regras contextuais que seriam utilizadas nas aplicações, a CRITiCAL necessita de uma DSL que fosse capaz de defini-las. Para tanto, foi desenvolvido a ContextRuleML, que realiza a modelagem das informações e regras contextuais, além de cumprir as características apresentadas na Subseção 3.2.2. O metamodelo da ContextRuleML é mostrado na Figura 4.2.

Das características avaliadas na Subseção 3.2.2, duas estiveram entre os menos alcançados pelos trabalhos analisados: o suporte à adaptação dinâmica do contexto, ausente em 80% dos trabalhos, e a independência de plataforma de *middleware*, que ocorre em apenas uma das abordagens. No entanto, dado o grau de relevância de ambos, foi decidido que eles seriam as características que teriam alta prioridade durante a criação da linguagem.

16 http://eclipse.org/
17 http://www.eclipse.org/modeling/emf/
18 http://www.eclipse.org/epsilon/doc/eugenia/

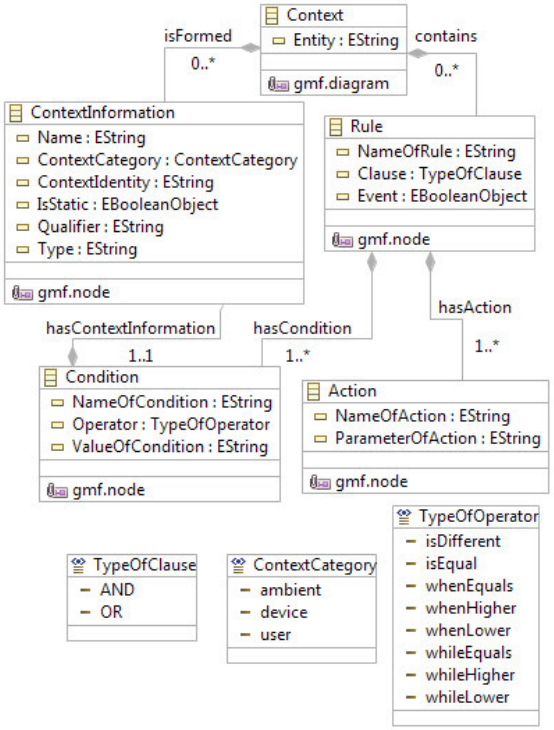


**Figura 4.2 –Metamodelo da ContextRuleML**

Para prover a independência da plataforma de *middleware* ao menos em um nível do modelo, foi decidido não inserir no modelo obtido pela DSL nenhum elemento de sintaxe de nenhuma plataforma de *middleware*. O metamodelo da ContextRuleML descreve o contexto e as regras contextuais de um nível de abstração maior. No caso da ferramenta CRITiCAL, os elementos de sintaxe específicos do LoCCAM estariam contidos nas regras de transformações. Essas regras que seriam responsáveis em traduzir essa sintaxe mais geral para a sintaxe mais específica do LoCCAM, realizando as adaptações necessárias de acordo com que o *middleware* provê.

Essa independência de plataforma do metamodelo também permite a reusabilidade dos modelos gerados através da DSL. Um mesmo modelo tem que ser passível de ser utilizado para diversas plataformas de middleware. Isso só é possível se forem definidas as regras de transformações para esta determinada plataforma.

A ContextRuleML foi baseada no metamodelo descrito pelo MLContext, voltado para modelagem de contexto permitindo a reusabilidade dos modelos [HOYOS, MOLINA e BLAYA, 2013]. Todavia, há uma adaptação dos elementos presentes nessa DSL para o objetivo do ContextRuleML e visando a modelagem visual das informações contextuais. Desta forma, ocorreu uma simplificação dos elementos constituintes do modelo da MLContext, mantendo-se apenas o estritamente essencial ao ContextRuleML.

Além disso, a ML Context não apresentava um suporte à representação de regras contextuais. Desta maneira, foi acrescida uma parte voltada para a definição das regras contextuais. Os elementos das regras contextuais foram baseados em um modelo mais geral apresentado pela abordagem de Santos *et al* [2013]. A DSL de Santos *et al* é voltada para o suporte de representação de regras e adaptação dinâmica do contexto.

Embora seja parte integrante do CRITiCAL, o ContextRuleML é uma DSL independente de plataforma de *middleware*. Através dela, é possível realizar a modelagem de regras contextuais e informações contextuais para outras plataformas de middleware. Para possibilitar isso, é necessário definir as regras de transformação equivalentes para essas plataformas. Ferramentas similares à CRITiCAL, mas que utilizem outro *middleware* para a aquisição e inferência das informações contextuais, podem ser criadas com base do ContextRuleML.

### 4.2.1 Visão Geral do ContextRuleML

A principal metaclasse do metamodelo da ContextRuleML é o Contexto (metaclasse *Context*). O Contexto é definido pela Entidade a qual ele pertence e é único para cada Entidade. O Contexto é formado por uma agregação de Informações Contextuais (metaclasse *ContextInformation*) e Regras (metaclasse *Rules*).

As Informações Contextuais são a parte principal da composição do Contexto. Elas são definidas por um conjunto de atributos, dentre os quais, destacam-se o seu nome e a categoria a qual esta informação contextual pertence. Exemplos de categorias podem ser as informações contextuais relacionadas com o ambiente, com o dispositivo, etc.

As Regras são compostas por uma agregação de Condições (metaclasse *Condition*) e Ações (metaclasse *Action*). Cada regra precisa necessariamente ter, pelo menos, uma condição e uma ação. No entanto, cada regra pode ter mais de uma condição e de uma ação. Nesse caso, todas as ações definidas na regra serão executadas apenas quando a composição das condições for satisfeita. As composições de condições na ferramenta podem ser feitas na forma de expressões lógicas com operadores “OR” e “AND”.

Cada Condição é formada pela Informação Contextual que deverá ser condicionada, o operador que indica o tipo de condição e o valor que quantifica a ação em si. O operador é definido pelo *enum* Operator.

### 4.2.2 Sintaxe Concreta

Para possibilitar a criação dos modelos pela ferramenta visual, foi definida uma sintaxe concreta do metamodelo apresentado na Figura 4.2. Através deste, é possível definir o que cada atributo do metamodelo corresponde na ferramenta visual e, posteriormente, para as transformações e gerações de código.

A sintaxe da metaclasse *Context* pode ser definida pela relação entre contexto e entidade. Existe um contexto, que é pertencente a uma Entidade (definida pelo atributo *Entity*). Para fins do objetivo da CRITiCAL, foi definido na ferramenta que, na modelagem realizada por ela, cada aplicação é uma Entidade. Deste modo, inicialmente há um único Contexto global para cada aplicação.

#### 4.2.2.1 Informações Contextuais

A metaclasse *ContextInformation* corresponde às Informações Contextuais que fazem parte do contexto da aplicação. Cada Informação Contextual corresponde a uma instância diferente da metaclasse, permitindo assim que várias informações contextuais façam parte de um mesmo contexto. Dentre os atributos da *ContextInformation*, pode-se mencionar: *Name*, *ContextCategory*, *ContextIdentity*, *Qualifier*, *isStatic* e *Type*.

Os atributos *Name* e *ContextIdentity* são aparentemente similares, mas tem uma diferença conceitual um para o outro. *Name* define o nome da informação contextual da forma como o desenvolvedor quer esta esteja nomeada na aplicação (e.g. “localTemperature”, “temp”, “tempAmbient”). *ContextIdentity* define a informação contextual da maneira como o middleware a interpreta (e.g., para o *middleware* LoCCAM, seria “temperature”, uma vez que seria o valor utilizado na composição das *ContextKeys* equivalentes).

O atributo *ContextCategory* define a categoria da informação contextual que está sendo criada (e.g. “ambient”, “device”) servindo para distinguir a natureza das informações contextuais que estão sendo modeladas. Uma categoria é definida pelo tipo de fonte de origem da informação contextual. Como exemplo, uma informação contextual relacionada à temperatura pode ser “ambient” caso esteja relacionada à temperatura na região em que o usuário se encontra (e.g. aplicações que desliguem o ar-condicionado caso a temperatura fique abaixo de um determinado valor), “device” caso esteja relacionada à temperatura do dispositivo naquele instante (e.g., aplicações que monitoram o dispositivo para evitar sobreaquecimento), ou “user” caso esteja relacionada à temperatura do usuário do

dispositivo(e.g., aplicações de monitoramento de pacientes enfermos com febre alta).

*Qualifier* é um atributo optativo que define alguma característica, qualificador ou requisito não-funcional da informação contextual. Pode-se usar como exemplo “kelvin” quando se deseja o valor da temperatura na escala Kelvin ao invés da Celsius. Ou “axisx” quando se deseja apenas o valor da aceleração no eixo x.

O atributo *isStatic* especifica se a informação contextual é estática ou dinâmica, ou seja, se ela mudaria conforme o contexto da aplicação. Um exemplo de informação contextual estática seria o nome de um usuário ou o nome de contatos ou amigos de um usuário. Na CRITiCAL, este atributo tem como valor padrão “falso”, indicando que são informações contextuais dinâmicas. Porém o desenvolvedor pode alterar este valor, se necessário.

O atributo *Type* define o tipo de retorno que o desenvolvedor espera para a informação contextual. Através dele, o desenvolvedor especifica se deseja que o valor da informação contextual seja um tipo especifico. Na CRITiCAL, o atributo *Type* é utilizado para especificar principalmente tipos primitivos do Java (e.g., int, double). Contudo, outros tipos podem ser utilizados se necessário. Dessa maneira, um desenvolvedor pode especificar se ele deseja que, por exemplo, uma informação contextual ambiente (e.g. temperatura, luminosidade, umidade relativa do ar) deva ser tratada como um double ou como um int.

#### 4.2.2.2 Regras, Condições e Ações

Dois dos principais objetivos da ContextRuleML são permitir o reuso dos modelos gerados e suportar a representação de regras contextuais. Assim sendo, a ContextRuleML requer uma sintaxe que seja o mais geral possível no tocante à representação de regras, desvinculando-se de qualquer plataforma de *middleware*. Desse modo, permite-se que um mesmo modelo possa gerar código para diversas plataformas, dependendo apenas da existência de regras de transformação equivalentes para tais plataformas.Assim sendo, a sintaxe proposta para as regras contextuais é composta de três metaclasses principais: *Rule*, *Condition* e *Action*.

*Rule* é a metaclasse principal e indica as regras contextuais que serão criadas no modelo. Cada regra tem, pelo menos, uma condição e uma ação. Na metaclasse *Rule*, há três atributos: *NameOfRule*, *isEvent* e *Clause*. O atributo *NameOfRule* define o nome da regra. O atributo *isEvent* determina se as regras serão definidas como um evento (no qual, uma regra/evento ativa uma determinada ação quando uma condição é satisfeita) ou como um

estado (em que a uma regra/estado ativa uma determinada ação enquanto uma condição é satisfeita). E o atributo *Clause* especifica como deverão ser a composição das condições. Esta composição pode ser determinada pelos operadores de expressões lógicas "OR" e "AND".

A definição das condições que compõem uma regra é realizada na metaclasse *Condition*. A sintaxe desta metaclasse define uma Condição, que é composta por um nome para a condição (atributo *NameOfCondition*), uma informação contextual (obtida da metaclasse *ContextualInformation*), um valor (atributo *Value*) e um operador (atributo *Operator*). O valor é utilizado como um comparativo com o valor recebido da informação contextual. O operador especifica qual a comparação a ser realizada. Ele é definido por um *enum Operator* que fornece oito situações possíveis (*WhenHigher*, *WhenLower*, *WhenEquals*, *WhileHigher*, *WhileLower*, *WhileEquals*, *IsEqual*, *IsDifferent*). As três primeiras especificam um evento, enquanto as cinco últimas indicam um estado.

A metaclasse *Action* especifica a ação a ser realizada pelo evento ou estado quando determinadas condições são satisfeitas. Esta ação é especificada pelos atributos *NameOfAction e ParameterOfAction*. O atributo *NameOfAction* corresponde à ação especificada. Esta ação deve estar pré-determinada nas regras de transformação. Quando não existe nenhuma regra para a construção de uma ação correspondente ao valor do atributo *NameOfAction* é gerado um esqueleto de código a ser preenchido pelo desenvolvedor após a transformação. O atributo *ParameterOfAction* permite a definição de algum parâmetro de entrada para esta ação. Inicialmente, as regras de transformação permitem o suporte para apenas um parâmetro, mas pretende-se aperfeiçoá-las futuramente para permitir vários parâmetros de entrada para uma ação. Essa metaclasse foi definida dessa maneira mais "livre" para permitir que novas ações sejam gradativamente inseridas nas regras de transformação sem que seja necessária realizar uma mudança no metamodelo ou na ferramenta visual.

### 4.2.3 Comparativo com as Características

A Tabela 4.1 posiciona a CRITiCAL em relação aos trabalhos relacionados no que se refere às características apresentadas na Subseção 3.2.2. A CRITiCAL e sua DSL, a ContextRuleML, foram projetadas de maneira a conseguirem satisfazer o maior número possível das características levantadas. Pois, graças a isso, pode-se garantir uma linguagem que conseguisse representar as regras contextuais de forma a auxiliar no reuso e manuntenabilidade dos modelos criados. Parte das características é cumprida pela DSL e parte delas pela ferramenta.

A representação de regras, a transparência, o nível independente de plataforma e a possibilidade de reuso dos modelos foram possibilitados pela forma como a ContextRuleML foi projetada. Ela foi pretendida como sendo uma linguagem que conseguisse expressar as informações e regras contextuais da forma simples e alto nível. Assim, foram evitadas quaisquer informações específicas da arquitetura interna do LoCCAM ou de qualquer outra plataforma de *middleware* nela. Isso possibilita não apenas que a CRITiCAL possa reutilizar modelos previamente criados, como também que uma ferramenta que utilize a ContextRuleML mas focando-se em outro *middleware* poderia reusar os modelos criados na CRITiCAL.

O suporte à adaptação, a ferramenta de transformação e a geração de artefatos são provido pela CRITiCAL. A primeira característica é possibilitada pela utilização do LoCCAM, enquanto as demais características são providas graças às transformações realizadas pela CRITiCAL.

A ContextRuleML é uma DSL independente de plataforma de *middleware*. Para que ela possa gerar artefatos de código voltados para outras plataformas de *middleware*, é necessário implementar as regras de transformação que traduzam os modelos criados por ela para as demais plataformas. Apesar disso, a CRITiCAL não é uma ferramenta independente de plataforma, uma vez que apesar de usar uma DSL com essa característica, ela é fortemente acoplada ao LoCCAM. Isso ocorre porque todas as características específicas da arquitetura interna do *middleware* LoCCAM, e que permitem a geração de código específico e compatível com o LoCCAM, estão presentes nos *templates* das regras de transformação e não na DSL.

**Tabela 4.1 – Comparação da CRITiCAL com os Trabalhos Relacionados em relação as Características desejadas**

| Caracterís-ticas | DSLs | | | | | |
|---|---|---|---|---|---|---|
| | **Abordagem de Almeida *et al*** | **Abordagem de Daniele *et al*** | **Copal-DSL** | **MLContext** | **Abordagem de Santos *et al*** | **CRITiCAL** |
| Independência de Plataforma de Middleware | Não (A-MUSE) | Não (A-MUSE) | Não (COPAL) | Sim | Não (plataforma de middleware proprietária) | Não (LoCCAM) |
| Tipos de Contexto | Sim | Sim | Sim | Sim | Não | Sim |
| Reuso de Modelos | Parcial | Parcial | Não | Sim | Não | Sim |
| Transparência | Sim | Sim | Sim | Sim | Sim | Sim |
| Suporte à Adaptação | Não | Não | Não | Não | Sim | Sim |
| Representação de Regras | Parcial | Parcial | Não | Não | Sim | Sim |
| Ferramenta de Transformação | GReAT, uma ferramenta baseada em GME | Medini QVT | XPand | MOFScript | - | XPand |
| Nível Independente de Plataforma | Sim | Sim | Não | Sim | Não | Sim |
| Geração de Artefatos | Uma especificação de um serviço de contexto dependente de plataforma | Uma especificação de uma aplicação móvel e sensível a contexto | *Bundles* OSGi | Esqueleto de código Java para inicializaçã o e instâncias contextuais | - | Um Projeto Android com métodos de comunicação entre o aplicativo e o *middleware* de gerenciamento de contexto |

## 4.3 Evolução do LoCCAM

Durante o processo de desenvolvimento do CRITiCAL, o *middleware* LoCCAM continuou sendo evoluído. O amadurecimento do *middleware* teve diversas motivações: a manutenção da compatibilidade do LoCCAM com as versões recentes do Android, a simplificação do processo de desenvolvimento de aplicações que utilizem o *middleware*, uma maior distinção dentre as informações contextuais que podem ser inferidas ou adquiridas pelo *middleware* e a necessidade de maior automatização e transparência na utilização do *middleware*.

Entre as principais modificações realizadas no *middleware* pode-se mencionar uma modificação no *middleware* para torná-lo transparente ao usuário final, o desenvolvimento da biblioteca LoCCAM_Lib, a criação de diversos novos CACs, junto com um repositório *on-*

*line* para armazenamento destes, e uma especificação maior da Árvore de Hierarquia utilizada. Por fim, foi desenvolvido um processo de adaptação transparente em tempo de execução, para quando uma determinada aplicação precisar de um CAC que não tenha sido previamente instalado no dispositivo móvel. Todas estas evoluções serão mais bem detalhadas nas Sub-Seções a seguir.

### 4.3.1 Transparência

Um dos principais empecilhos encontrados para um maior uso do LoCCAM era a ausência de transparência do *middleware* para o usuário final. No dispositivo móvel, o LoCCAM era um aplicativo que tinha que ser ativado pelo usuário antes que este pudesse utilizar as aplicações que dependiam dele, além de precisar instalar manualmente cada um dos CACs necessários em um diretório específico do dispositivo.

Com esta evidente limitação, buscou-se tornar o LoCCAM transparente ao usuário final. A primeira etapa do processo de maior transparência do LoCCAM foi obtida realizando uma modificação no processo de ativação do *middleware*. O LoCCAM deixou de ser um aplicativo visível ao usuário e passou a funcionar integralmente como um serviço. Esse serviço é ativado quando a primeira aplicação que dependa do *middleware* seja inicializada e é desativado quando a última aplicação que o esteja utilizando seja encerrada. O único momento em que o usuário final tem ciência da existência do *middleware* é durante o processo de instalação deste no dispositivo móvel.

A segunda etapa, correspondente à automatização da instalação dos CACs no dispositivo móvel será mais bem descrita na Subseção 4.3.4.

### 4.3.2 Representação do Modelo de Contexto

Durante o processo de criação dos novos Componentes de Aquisição de Contexto, foi percebido que a Árvore de Hierarquia mostrada na Figura 2.8 não conseguia abranger satisfatoriamente as informações contextuais adquiridas. Muitas vezes era necessário que o desenvolvedor realizasse um tratamento na informação contextual adquirida pelo CAC para que esta correspondesse ao que a aplicação realmente precisava.

Uma das razões para que esse problema acontecesse era devido ao fato dos CACs retornarem os valores das informações contextuais na escala utilizada pelos sensores do Android, que normalmente correspondem à escala do Sistema Internacional. No entanto, em determinadas aplicações era necessário que a informação contextual estivesse em outra escala.

Um exemplo desse problema seria a informação contextual para Temperatura Ambiental, cuja *Context Key* é “context.ambient.temperature”, retornar o valor da temperatura em Celsius e a aplicação necessitar dos valores em Fahrenheit. Isso dificultava até mesmo a criação de filtros contextuais.

Para solucionar tal problema foi proposta uma modificação na Árvore de Hierarquia, com a adição de uma nova ramificação. Esta ramificação seria optativa e corresponderia a qualificadores. Esses qualificadores seriam requisitos não-funcionais ou especificações da informação contextual mais geral. A Figura 4.3 exemplifica a nova Árvore de Hierarquia. As *Context Keys* podem ser compostas por três ou quatro nós da Árvore. Quando compostas por três nós, elas correspondem à informação contextual em seu valor "padrão". Quando compostas por quatro nós, elas correspondem à informação contextual de uma forma mais específica. Esse quarto nó poderia ser uma escala diferente, um eixo específico, um *timestamp* diferente do padrão, dentro outros. No exemplo da Figura 4.3, a *Context Key* “context.ambient.temperature” retornaria o valor da temperatura ambiental em Celsius, enquanto a *Context Key* “context.ambient.temperature.kelvin” retornaria o valor em Kelvin.

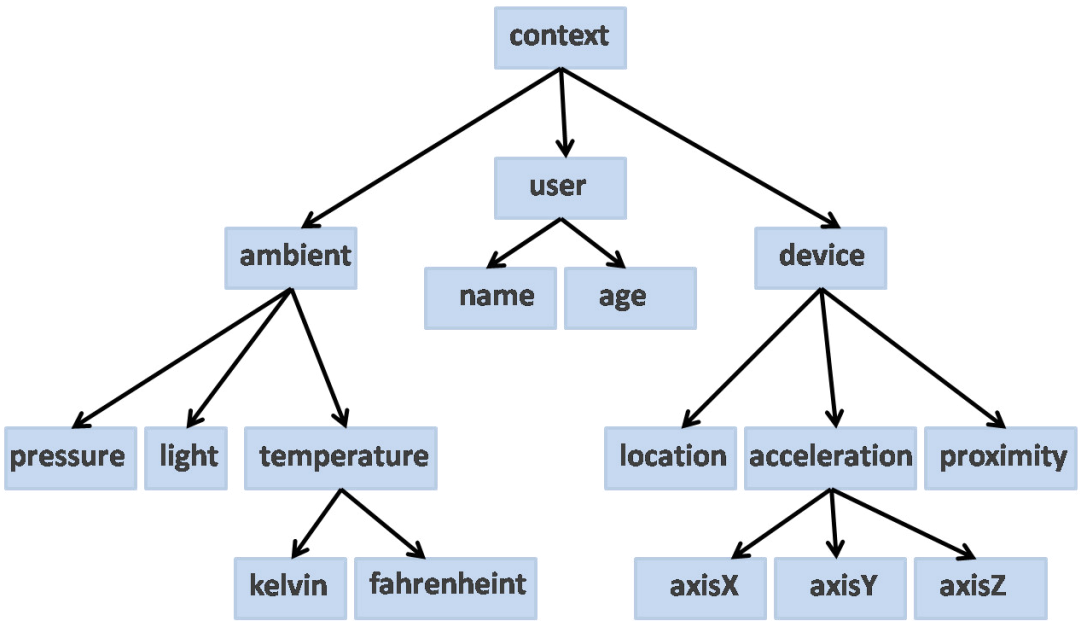


**Figura 4.3 – Nova Arvore de Hierarquia**

A Tabela 4.2 mostra alguns exemplos de qualificadores que podem ser criados com base em uma determinada informação contextual.

**Tabela 4.2 – Exemplo de Qualificadores**

| Informação Contextual | Qualificador | Descrição |
|---|---|---|
| Temperature | Kelvin | Retorna o valor da temperatura ambiente na escala Kelvin |
| | Fahrenheit | Retorna o valor da temperatura ambiente na escala Fahrenheit |
| Acceleration | AxisX | Retorna o valor do módulo da aceleração do dispositivo móvel no eixo X |
| | AxisY | Retorna o valor do módulo da aceleração do dispositivo móvel no eixo Y |
| | AxisZ | Retorna o valor do módulo da aceleração do dispositivo móvel no eixo Z |
| Random | OneSecond | Retorna um número randômico a cada um segundo |
| | FiveSecond | Retorna um número randômico a cada cinco segundos |
| | Range20 | Retorna um número randômico entre 0 e 20 no *timestamp* padrão (dez segundos) |
| | Range12 | Retorna um número randômico entre 0 e 12 no *timestamp* padrão (dez segundos) |

### 4.3.3 Biblioteca LoCCAM_Lib

O código dos métodos tradicionais de comunicação entre a aplicação e o LoCCAM tinham algumas desvantagens inerentes à sua implementação. Eles eram complexos e repetitivos. Com o objetivo de simplificar essa comunicação e abstrair ao desenvolvedor boa parte da repetição do código, foi desenvolvida uma biblioteca para o LoCCAM, a LoCCAM_Lib.

A LoCCAM_Lib encapsula os métodos de comunicação entre a aplicação e o LoCCAM e entre a aplicação e o SysSU. Os métodos encapsulados compreendem os tratamentos das informações contextuais e a conexão/desconexão do elo entre *middleware* e aplicação. Dentre eles pode-se mencionar:

- Put: responsável por publicar o interesse da aplicação em determinada informação contextual;
- Read: realiza a leitura do valor de informações contextuais publicadas;
- Subscribe: indica que a aplicação deseja ser notificada quando uma informação contextual que for de seu interesse vier a ser publicada;
- Unsubscribe: indica que a aplicação não deseja mais ser notificada sobre uma determinada informação contextual;

- Take: retira o interesse da aplicação em determinada informação contextual.

Embora tenha atingido seu objetivo a contento, a LoCCAM_Lib não abstrai totalmente a sintaxe do *middleware*. Ela ainda requer que o desenvolvedor conheça os métodos a serem utilizados e a *ContextKey* de cada informação contextual.

Como a LoCCAM_Lib já realiza a uniformização da sintaxe do LoCCAM, a CRITiCAL utiliza essa biblioteca na construção do código gerado. Por exemplo, a LoCCAM_Lib fornece um método de leitura do valor da informação contextual. A CRITiCAL utiliza este método para gerar todos os métodos para a leitura de cada informação contextual indicadas pelo desenvolvedor no modelo.

### 4.3.4 Adaptação Dinâmica em Tempo de Execução

Uma desvantagem antiga do LoCCAM era a necessidade da instalação prévia dos CACs para que as aplicações pudessem funcionar. Isso, aliado ao fato de que a instalações de cada CAC era realizada de maneira manual e feita dispositivo a dispositivo, tornava a utilização de aplicações integradas ao LoCCAM uma tarefa bastante custosa em termos de tempo.

Desta forma, uma das características modificadas no LoCCAM foi a automatização e o aumento na transparência da instalação dos componentes. Assim, o LoCCAM consegue se adaptar dinamicamente em tempo de execução, caso haja a necessidade de um CAC não existente no dispositivo.

Quando uma aplicação publica o interesse em uma determinada informação contextual para a qual não há o CAC equivalente instalado no dispositivo, o LoCCAM recebe uma lista com as Context Keys dos CACs que não estão disponíveis no dispositivo móvel.

O LoCCAM faz a verificação da versão do Android presente no dispositivo e então faz um acesso ao Repositório de CACs. No repositório, o LoCCAM faz a busca dos CACs correspondentes e realiza a triagem para escolher os da versão compatível com o dispositivo. Uma vez encontrados, os CACs são baixados, instalados, ativados e começam a publicar as informações contextuais no SySSU. Isso ocorre em tempo de execução da aplicação.

Para possibilitar tal triagem, uma modificação foi realizada no Manifest de todos os CACs. Nesta modificação, foi adicionado um novo campo: Minimal-Version.

O Minimal-Version corresponde à versão mínima requerida do Sistema Operacional Android para que o CAC possa executar as suas funcionalidades. Isso possibilita que o

LoCCAM possa se adaptar de uma forma que permaneça compatível com a versão do Android que executa no dispositivo móvel, desde que exista um CAC que corresponda à informação contextual e exija uma versão do Android menor ou igual à existente no dispositivo móvel. A Figura 4.4 apresenta a nova versão do Manifest dos CACs.

DeviceLinearAccelerationAxisXCAC

```
1 Manifest-Version: 1
2 Built-By: Paulo
3 Context-Provided: context.device.linearacceleration.axisx
4 Bundle-Activator: br.ufc.cac.activator.Activator
5 Bundle-ManifestVersion: 2
6 Bundle-Name: br.ufc.cac.devicelinearaccelerationaxisx
7 Bundle-SymbolicName: DeviceLinearAccelerationAxisXCAC
8 Bundle-Version: 2.0.0
9 Minimal-Version: 2.3
10 Export-Package: br.ufc.cac.activator,br.ufc.cac.impl
11 Import-Package: android.content,android.util,android.hardware,
12  br.ufc.loccam.isensor,br.ufc.loccam.ipublisher,br.ufc.loccam.iandroidcon
```

**Figura 4.4 – Novo Manifest dos CACs**

No entanto, esta adaptação ocorre apenas para CACs que estejam no Repositório online. Assim, CACs que não tenham sido armazenados no Repositório, seja por serem CACs proprietários, em fase de testes ou para usos pessoais, ainda precisariam ser instalados previamente. No entanto, este processo é automatizado pela CRITiCAL. A lista de CACs existentes atualmente no Repositório está exibida no Apêndice A.

## 4.4 Ferramenta de Modelagem Visual

Através do metamodelo da ContextRuleML, foi desenvolvida uma ferramenta para modelagem visual das informações e regras contextuais. A Figura 4.5 mostra um exemplo da modelagem de uma regra contextual com duas condições e uma ação na ferramenta visual.

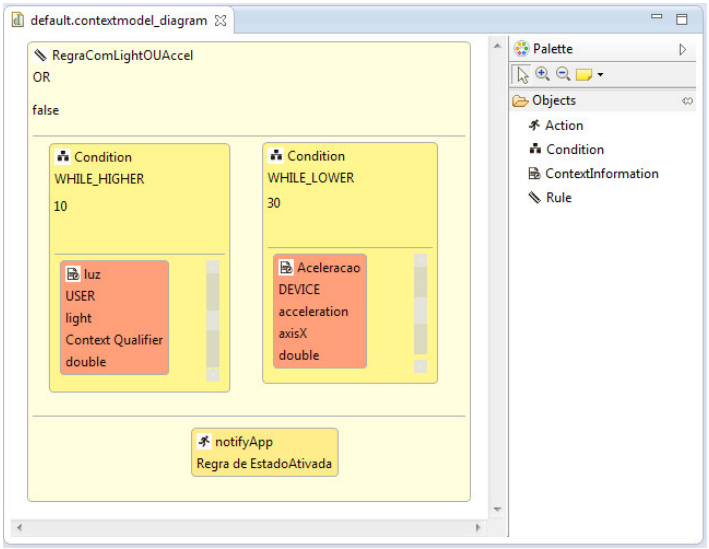


**Figura 4.5 – Ferramenta Visual**

A inserção de elementos no modelo segue o ContextRuleML. Ações e Condições só podem ser inseridas dentro de Regras. Informações Contextuais podem ser inseridas dentro de Condições ou no Contexto. Regras são inseridas no Contexto. A modelagem é realizada de forma simples: o desenvolvedor clica no ícone do elemento que ele deseja inserir e, então, clica na região do modelo em que ele deseja que o elemento seja inserido. Se a tentativa inserção do elemento ocorrer em uma região proibida pelo metamodelo, a ferramenta não deixa que o desenvolvedor a insira. Assim, pretende-se que a ferramenta deixe a modelagem das informações e regras o mais simples e intuitivo quanto possível.

Os atributos podem ser editados tanto diretamente clicando em cima das caixas correspondentes no gráfico, quanto em uma caixa de dialogo realizada na parte inferior da tela.

Uma vez realizada a modelagem, ela é salva como um arquivo de extensão .contextmodel. O arquivo ContextModel contém uma correspondência à representação visual do modelo expressa pelo desenvolvedor no formato de um arquivo XMI. Um exemplo de arquivo .contextmodel é apresentado na Figura 4.6. Esse arquivo que será utilizado pela CRITiCAL para a geração do código e a configuração do middleware no dispositivo móvel.

model.contextmodel

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <metamodel:Context xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:metamodel="http://www.example.org/metamodel">
3   <contains NameOfRule="RuleLight" Event="false">
4     <hasCondition Operator="whileLower" ValueOfCondition="3">
5       <hasContextInformation Name="LightLoccam" ContextIdentity="light" Qualifier=" " Type="double"/>
6     </hasCondition>
7     <hasAction NameOfAction="NotifyApp" ParameterOfAction="Activate"/>
8   </contains>
9 </metamodel:Context>
```

**Figura 4.6 – Exemplo do código de um arquivo .contextmodel**

## 4.5 Geração de Código

Uma vez o desenvolvedor tenha projeto o modelo desejado, com todas as regras e informações contextuais que ele deseja inserir na aplicação, ele pode iniciar uma das duas transformações que o CRITiCAL permite. A primeira transformação corresponde à geração do código Java equivalente ao modelo. O produto final desta transformação é um projeto Android, com todos os métodos de comunicação com o LoCCAM.

### 4.6.1 Processo de Geração de Código

O processo de geração de código é realizado com base no modelo criado pelo desenvolvedor e no *template* das regras de transformações equivalente ao da geração de

regras. Este *template* será mais bem descrito na Subseção 4.6.2.

A transformação é realizada através das seguintes etapas:

1. **Geração do projeto Android:** É gerado um projeto Android voltado para a IDE Eclipse. Inicialmente neste projeto já estão inseridos a biblioteca Loccam_Lib e dois pacotes. No primeiro deles, será inserido um esqueleto de uma *Activity* principal para o desenvolvedor poder iniciar a sua aplicação. O outro pacote está a priori vazio, uma vez que nele será inserida a classe loccamActivator;
2. **Criação da classe LoccamActivator:** A classe LoccamActivator é criada com os métodos de conexão e desconexão da comunicação com o *middleware*;
3. **Definição das *ContextKeys* e dos métodos de gerenciamento das informações contextuais:** As *Context Keys* são montadas usando os atributos *ContextCategory*,*ContextIdentity* e *Qualifier* das informações contextuais. São criados métodos específicos para o gerenciamento transparente de cada informação contextual;
4. **Criação dos métodos de gerenciamento das informações contextuais:** Uma vez que as *Context Keys* tenham sido instanciadas, são criados métodos específicos para cada informação contextual, provendo as ações de *put*, *take* e *read* de maneira transparente;
5. **Criação dos filtros contextuais e das ações:** As condições expressas no modelo são traduzidas como filtros do SysSU. As ações definidas no modelo são criadas e chamadas nos *reactions* do *middleware*. No LoCCAM, cada subscrição recebe apenas um filtro e um *reaction*. Assim, quando uma há uma composição de condições, esta composição é realizada dentro do *reaction*;
6. **Criação dos métodos que realizam a subscrição e desubscrição:** As regras do modelo são traduzidas como métodos para subscrição e desubscrição de filtros no LoCCAM. Eles são os equivalentes no modelo à metaclasse *Rules*. Estes métodos recebem o *reaction*, a informação contextual e a condição (*filtres*) para criar a subscrição. O método criado, chamado *listenCondition*, recebe um parâmetro booleano como entrada. Caso este seja verdadeiro, a subscrição é realizada. Caso seja falso, a desubscrição é feita;
7. **Criação da *Activity* Principal:** Uma *Activity* principal é criada já com os métodos de comunicação com o LoCCAM gerados e ativados. Além disso, o código gerado já

realizar a instanciação da classe LoccamActivator para o desenvolvedor. Dessa maneira, obtém-se uma maior abstração do *middleware* para o desenvolvedor.

Um *workflow* do CRITiCAL, da modelagem até a aplicação sendo executada, é mostrado na Figura 4.7. Este *workflow* ilustra um exemplo de uso da ferramenta na qual foi criada uma aplicação simples. Esta aplicação faz a leitura da luminosidade e muda a cor da imagem para verde, quando a luminosidade fica abaixo de um determinado valor.

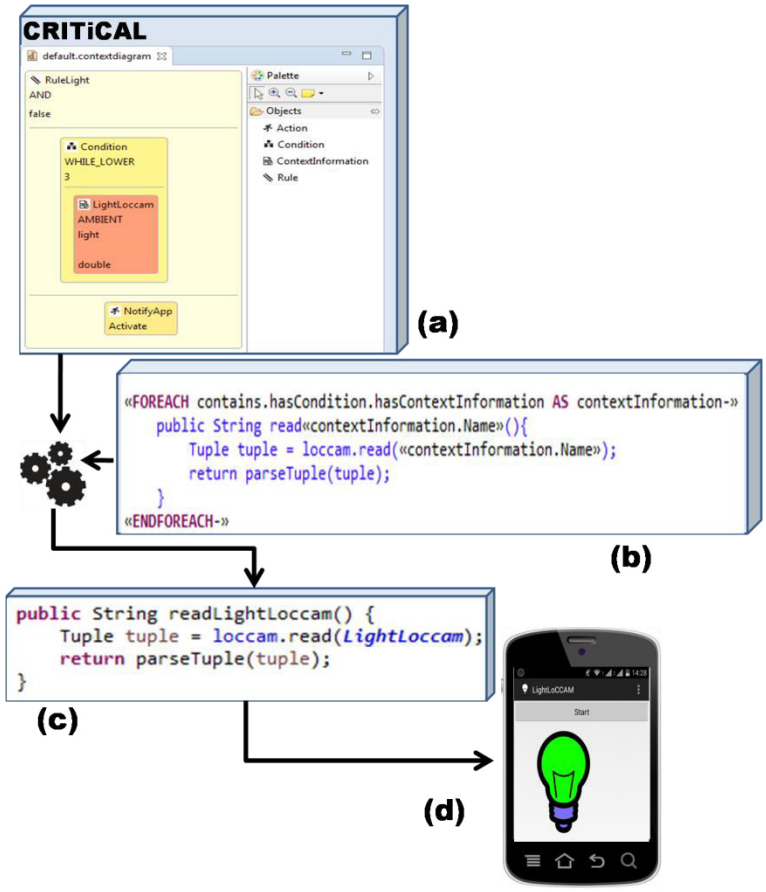


**Figura 4.7 – Processo de Geração do Código**

Na Figura 4.7(a) é realizada a modelagem visual da regra contextual desejada, especificando a condição e a informação contextual esperada. No caso, se fosse necessário, também seriam definidos os qualificadores da informação contextual. Na Figura 4.7(b) é mostrado os templates de regras de transformação que especificam como a modelagem realizada deve ser traduzida para a sintaxe que o LoCCAM possa utilizar. Estes templates são transparentes ao desenvolvedor, que não precisa criá-los. Com base no modelo da Figura 4.7(a) e no *template* da Figura 4.7(b) é realizada a transformação e consequente geração do código. Na Figura 4.7(c) é mostrado um exemplo do código gerado através da Figura 4.7(a) e Figura 4.7(b).

O produto final da transformação é um projeto Android, que tem ambas as classes geradas e a biblioteca LoCCAM_Lib. Com base neste projeto Android, a aplicação pode ser continuada, agora com o acesso simples e transparente às informações e regras contextuais já criadas, podendo ser utilizadas quando forem necessárias. Na Figura 4.7(d) é mostrada a aplicação Android desenvolvida com base na modelagem realizada na Figura 4.7(a) e utilizando o código gerado ma Figura 4.7(c).

### 4.5.2 *Templates* de Geração

Para realizar a transformação do modelo especificado no arquivo .contextmodel para as classes Java correspondentes, fez-se necessário a construção de *templates* implementados em XPand. Esses *templates* encapsulam as regras de transformação voltadas a criação das classes *MainActivity* e *LoccamActivator*, realizando as etapas 2 a 7 do processo de geração do código especificado na Subseção 4.6.1.

A Figura 4.8 mostra um trecho do *template*, com as regras para a criação de métodos de leitura da informação contextual. No caso exemplificado na Figura 4.8, é mostrado que o *template* realiza um laço buscando todas as informações contextuais contidas no Contexto e na Condição e colocando os seus atributos em um método próprio e padronizado. Para a criação de filtros, *reactions* e subscrições, são realizados laços similares com as Condições e Regras contextuais especificadas no modelo.

```
«REM»Criação dos métodos de Read«ENDREM»
«FOREACH isFormed AS contextInformation-»

    /**
     * Nome: read«contextInformation.Name»
     * Entrada: -
     * Saída: String com o valor da informação contextual «contextInformation.Name»
     * Descrição: Realiza a leitura da informação contextual «contextInformation.Name»
     *              e retorna o valor da leitura como uma String
     */
    public String read«contextInformation.Name»(){
        Tuple tuple = null;
        try {
            Tuple tuple = loccam.read(«contextInformation.Name»);
            } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
            return parseTuple(tuple);
    }
«ENDFOREACH-»

«FOREACH contains.hasCondition.hasContextInformation AS contextInformation-»

    /**
     * Nome: read«contextInformation.Name»
     * Entrada: -
     * Saída: String com o valor da informação contextual «contextInformation.Name»
     * Descrição: Realiza a leitura da informação contextual «contextInformation.Name»
     *              e retorna o valor da leitura como uma String
     */
    public String read«contextInformation.Name»(){
        Tuple tuple = null;
        try {
        Tuple tuple = loccam.read(«contextInformation.Name»);
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return parseTuple(tuple);
    }
«ENDFOREACH-»
```

**Figura 4.8 – Exemplo do *Template* do Código**

### 4.5.3 Produto Final da Transformação

No final do processo de transformação, o desenvolvedor obtém um projeto Android com a estrutura exibida na Figura 4.9. O projeto, nomeado TemplateAndroidProject, corresponde a um projeto Android normal com a adição de uma biblioteca (LoCCAM_Lib) e dois pacotes:

- **activities:** Onde está inserida a classe MainActivity gerada.
- **loccam:** Onde está inserida a classe LoccamActivator gerada.

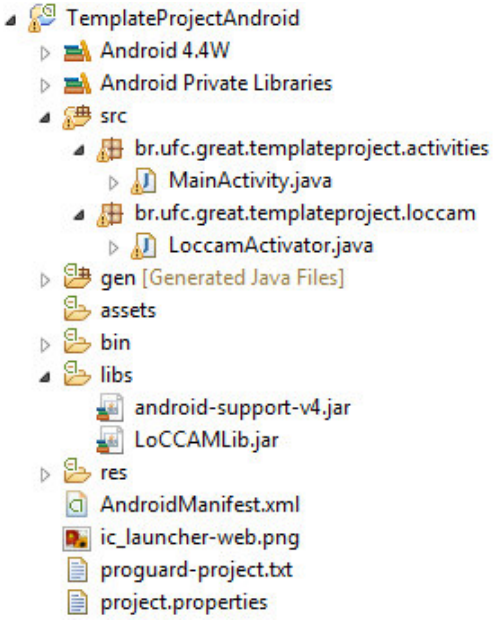


**Figura 4.9 – Estrutura do Projeto Android gerado**

Caso nenhum erro tenha acontecido durante o processo de modelagem, o projeto não deve registrar nenhum erro de compilação. Desta maneira, o desenvolvedor pode iniciar o desenvolvimento da aplicação sensível a contexto. Nas subseções a seguir, são detalhadas as estruturas das duas classes geradas.

#### 4.5.3.1 MainActivity

A *MainActivity* gerada pelo CRITiCAL é uma *Activity* que contém o mínimo de métodos necessários. Basicamente, ela se resume aos mesmos métodos que já são criados automaticamente em uma *Activity* padrão, mas com a adição de elementos de ligação com a classe *LoccamActivator*.

Essas adições, mostradas na Figura 4.10, correspondem à instanciação da *LoccamActivator*, o início da comunicação entre aplicação e o *middleware*, o método para adição de CACs não instalados no dispositivo móvel, e a desconexão da comunicação com o LoCCAM quando a aplicação for encerrada.

Como a classe *LoccamActivator* já está instanciada, na *MainActivity* já se é possível realizar as chamadas aos métodos da LoccamActivator. Por exemplo, no caso de um modelo que tivesse uma informação contextual de nome "Light" e uma regra contextual de nome "FilterLight", as chamadas aos métodos da LoccamActivator seriam da seguinte forma:

- **Publicação de Interesse Contextual:** loccam.putLightInterest();
- **Retirada de Interesse Contextual:** loccam.putLightInterest();

- **Leitura de Interesse Contextual:** String valorLido = loccam.readLight();
- **Subscrição de Regra Contextual:** loccam.listenConditionFilterLight(true);
- **Desubscrição de Regra Contextual:** loccam.listenConditionFilterLight(false).

MainActivity.java

```
package br.ufc.great.templateproject.activities;

import br.ufc.great.templateproject.loccam.LoccamActivator;

public class MainActivity extends Activity implements LoccamListener {

    private LoccamActivator loccam;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Instancia o LoccamActivator
        loccam = LoccamActivator.getInstance(getApplicationContext(), getPackageName());
        // Inicializa a comunicação com o LoCCAM
        loccam.connect(this);
    }

    @Override
    public void onServiceConnected(ISysSUService arg0) {
        // TODO Auto-generated method stub
        // Publica o interesse da aplicação pelas informações contextuais
        loccam.putCACs();
    }

    @Override
    public void onServiceDisconnected() {
        // TODO Auto-generated method stub
        loccam.disconnect();
    }

    @Override
    protected void onDestroy() {
        onServiceDisconnected();
        super.onDestroy();
    }
}
```

**Figura 4.10– Trecho de Código do Main Activity**

### 4.5.3.2 LoccamActivator

A classe *LoccamActivator* encapsula todos os métodos de comunicação entre a aplicação e o LoCCAM. O seu objetivo é funcionar como uma interface mediadora entre a aplicação e o LoCCAM. A Figura 4.11 mostra como deve é o fluxo de comunicação entre aplicação e o sistema operacional Android. O LoCCAM é responsável por intermediar o acesso às informações contextuais, sejam elas oriundas de sensores ou serviços remotos.

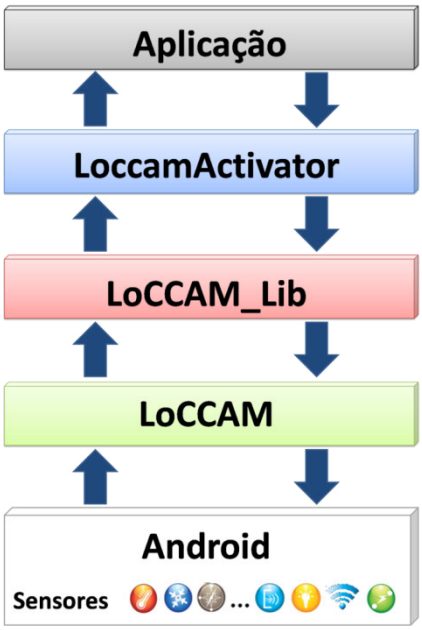


**Figura 4.11 – Comunicação entre LoCCAMActivator e os Sensores**

A LoCCAM_Lib fornece métodos que uniformizam e simplificam a sintaxe do *middleware*, tanto do uso das informações contextuais pela aplicação, quanto à comunicação entre aplicação e *middleware*. Por sua vez, a classe LoccamActivator, através da LoCCAM_Lib, instancia os métodos de comunicação com o *middleware*; os filtros, reactions e subscrições modeladas previamente pelo desenvolvedor; e os métodos para a comunicação e leitura de cada uma das informações contextuais indicadas pelo desenvolvedor no modelo.

Os métodos de leitura da informação contextual são mostrados na Figura 4.12. Para cada informação contextual, é realizada a chamada do método de leitura da biblioteca, recebendo o nome da informação contextual como parâmetro. Como o método da biblioteca retorna a tupla inteira, é realizada uma segunda etapa para se obter o valor da informação contextual de dentro da tupla.

```
/**
 * Nome: readLightLoCCAM
 * Entrada: -
 * Saída: String com o valor da informação contextual LightLoCCAM
 * Descrição: Realiza a leitura da informação contextual LightLoCCAM
 *          e retorna o valor da leitura
 */
public String readLightLoCCAM() {
    Tuple tuple = null;
    try {
        tuple = loccam.read(LightLoCCAM);
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return parseTuple(tuple);
}
```

**Figura 4.12– Trecho de Código do Read**

A Figura 4.13 exemplifica a chamada de uma regra compostas por três condições e que seja um evento. Nesse caso, quando a regra é ativada, são chamadas seis subscrições ao LoCCAM, sendo três correspondentes às condições especificadas e três subscrições que são o oposto das três originais. A subscrição original e a sua oposta ficam se alternando: A subscrição "oposta" inicia bloqueada e sempre que a condição de uma subscrição original é satisfeita, a original se bloqueia e libera a sua subscrição oposta. Quando a condição oposta é satisfeita, ela se bloqueia e libera a subscrição original. E, assim, sucessivamente. Com isso, é garantido que, em uma regra de evento, a aplicação seja notificada apenas quando o evento realmente acontecer.

```
/**
 * Nome: listenConditionRegraDeTemp
 * Entrada: Um booleano indicando se a regra deve ou não ser ativada
 * Saída: -
 * Descrição: Quando recebe um verdadeiro, ativa a regra RegraDeTemp
 *     e todas as subscrições às informações contextuais contidas nesta
 */
public void listenConditionRegraDeTemp(boolean command) {
    if (command) {

        RegraDeTempWithTempFahrenheit = loccam.subscribe(
                RegraDeTempWithTempFahrenheitReaction, "put",
                TempFahrenheit, RegraDeTempFilter);
        NoRuleRegraDeTempWithTempFahrenheit = loccam.subscribe(
                NoRuleRegraDeTempWithTempFahrenheitReaction, "put",
                TempFahrenheit, NoRuleRegraDeTempFilter);

        RegraDeTempWithTempKelvin = loccam.subscribe(
                RegraDeTempWithTempKelvinReaction, "put", TempKelvin,
                RegraDeTempFilter);
        NoRuleRegraDeTempWithTempKelvin = loccam.subscribe(
                NoRuleRegraDeTempWithTempKelvinReaction, "put", TempKelvin,
                NoRuleRegraDeTempFilter);

        RegraDeTempWithTempCelsius = loccam.subscribe(
                RegraDeTempWithTempCelsiusReaction, "put", TempCelsius,
                RegraDeTempFilter);
        NoRuleRegraDeTempWithTempCelsius = loccam.subscribe(
                NoRuleRegraDeTempWithTempCelsiusReaction, "put",
                TempCelsius, NoRuleRegraDeTempFilter);

    } else {
        if (RegraDeTempWithTempFahrenheit != null) {
            if (RegraDeTempWithTempKelvin != null) {
                if (RegraDeTempWithTempCelsius != null) {
                    loccam.unSubscribe(RegraDeTempWithTempFahrenheit);
                    loccam.unSubscribe(NoRuleRegraDeTempWithTempFahrenheit);
                    loccam.unSubscribe(RegraDeTempWithTempKelvin);
                    loccam.unSubscribe(NoRuleRegraDeTempWithTempKelvin);
                    loccam.unSubscribe(RegraDeTempWithTempCelsius);
                    loccam.unSubscribe(NoRuleRegraDeTempWithTempCelsius);
                }
            }
        }
    }
}
```

**Figura 4.13– Trecho de Código do Subscribe**

## 4.6 Configuração do *Middleware*

O CRITiCAL tem duas opções de transformação diferentes para um mesmo modelo. A primeira gera o código do projeto Android para o desenvolvimento de uma aplicação que utilize o LoCCAM. A segunda opção realiza a configuração do *middleware* em um dispositivo móvel. Ela automatiza a instalação do LoCCAM e dos CACs correspondentes às informações contextuais especificadas no modelo.

Uma vez que o desenvolvedor tenha criado o modelo, ele pode optar por essa transformação. Ela é gerenciada por um Instalador do CRITiCAL e, a partir do modelo, gera um arquivo chamado CACList. A partir do CACList, o Instalador começa o processo de seleção de CACs e instalação do *middleware* e dos componentes no dispositivos.

A Figura 4.14 ilustra o processo de configuração, com base no modelo mostrado na Figura 4.14(a). Uma vez que a modelagem tenha sido finalizada, como na Figura 4.14(a), e o modelo tenha sido salvo (sua extensão é ".contextmodel"), uma janela é aberta. Esta janela, mostrada na Figura 4.14(b), gerencia todo o processo de configuração. Esta janela pede para que o desenvolvedor selecione o modelo que ele deseja que seja utilizado para a configuração (e.g., o modelo recém salvo ou algum outro de extensão ".contextmodel") e qual o dispositivo móvel ou emulador que deve ser o alvo da instalação. Uma vez que ambos estejam definidos, é realizada a instalação do LoCCAM no dispositivo-alvo e sua posterior inicialização. O passo seguinte é uma transformação com base no modelo da Figura 4.14(a) para a geração de um arquivo de texto simples com a lista das *Context Keys* equivalentes às informações contextuais modeladas pelo desenvolvedor. Esta lista é mostrada na Figura 4.14(c). Com base nesta lista e no dispositivo-alvo escolhido, é verificado se o dispositivo tem os sensores necessários para a execução das aplicações. Em caso positivo, a janela pede ao desenvolvedor que indique em qual diretório estão os CACs que ele deseja instalar.

A ferramenta vasculha esse diretório e realiza a triagem dos CACs que sejam correspondentes às *Context Keys* definidas na lista. Quando encontradas, ela instala estes CACs nas pastas correspondentes. Caso algum CAC correspondente não seja encontrado no diretório especificado, a ferramenta realiza o download do CAC correspondente no repositório *on-line* de CACs. O processo de leitura, busca e instalação pode ser acompanhado via console pelo desenvolvedor, conforme mostrado na Figura 4.14(d).

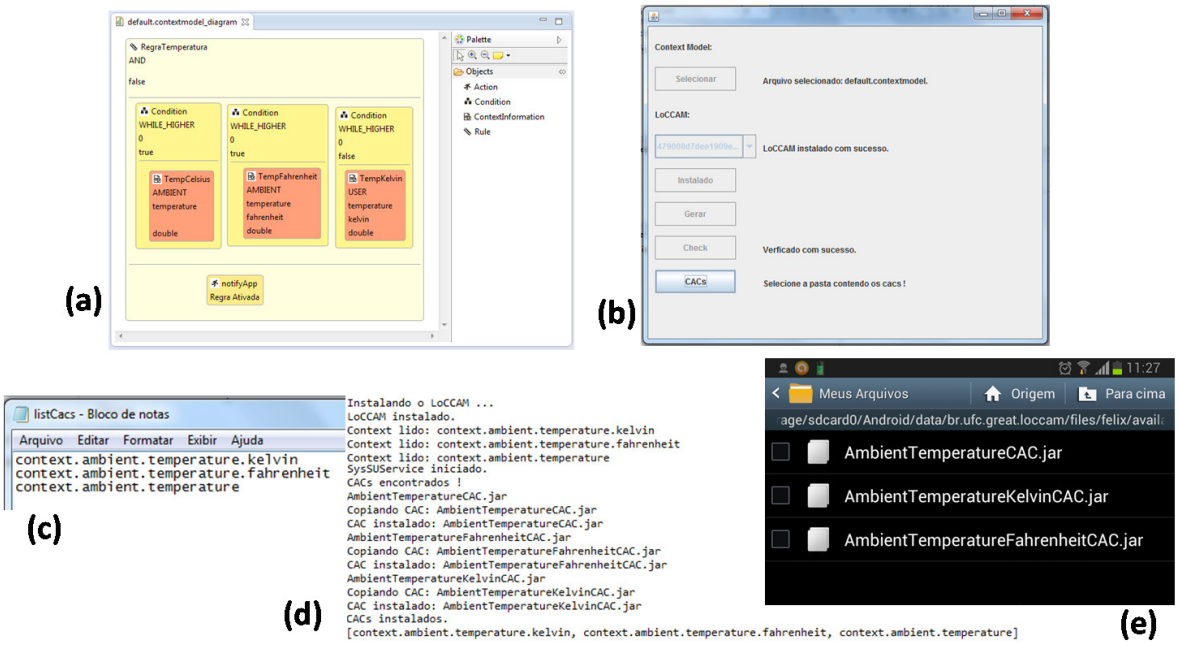


**Figura 4.14– Configuração automática do LoCCAM**

O processo de leitura, busca e instalação pode ser acompanhado via console pelo desenvolvedor. O tempo médio de execução de todo o processo é de 2117 ms, em um experimento realizado em um Pentium Dual Core 2 GHz, com 2 GB de memória RAM DDR2 e com sistema operacional Windows 7 32bits.

## 4.6.1 Geração de Lista de *Context Keys*

Durante o processo de configuração do *middleware* e instalação dos componentes, o modelo criado pelo desenvolvedor é utilizado para a geração do CACList. O CACList é um simples arquivo de texto contendo a lista das *Context Keys* equivalentes às informações contextuais modeladas pelo desenvolvedor.

Template.xpt

```
1 «IMPORT metamodel»
2
3 «DEFINE main FOR Context»
4 «FILE "CacsList.txt"-»
5 «FOREACH isFormed AS contextinformation-»
6 «IF contextinformation.Qualifier.toString().contains("null")-»
7 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()»
8 «ELSEIF contextinformation.Qualifier.toString().length<=1-»
9 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()»
10 «ELSE-»
11 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()».«contextinformation.Qualifier.toLowerCase()»
12 «ENDIF-»
13 «ENDFOREACH-»
14 «FOREACH contains.hasCondition.hasContextInformation AS contextinformation-»
15 «IF contextinformation.Qualifier.toString().contains("null")-»
16 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()»
17 «ELSEIF contextinformation.Qualifier.toString().length<=1-»
18 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()»
19 «ELSE-»
20 context.«contextinformation.ContextCategory».«contextinformation.ContextIdentity.toLowerCase()».«contextinformation.Qualifier.toLowerCase()»
21 «ENDIF-»
22 «ENDFOREACH-»
23 «ENDFILE»
24
25
26
27 «ENDDEFINE»
```

**Figura 4.15 – *Template* para geração do CACList**

Para poder gerar essa lista de CACs foi implementado um segundo *template* com as regras de transformação para xPand. Este *template* é mostrado na íntegra na Figura 4.15. Basicamente, a regra de transformação utilizada vasculha todas as Informações Contextuais contidas no Contexto ou nas Condições do Modelo e, então, monta a *Context Key* equivalente de acordo com os atributos fornecidos por esta.

### 4.6.2 Automatização da Instalação do *Middleware* e CACs

A Figura 4.16 mostra a interface do Instalador responsável pelo gerenciamento do processo de configuração do *middleware* através do CRITiCAL. Ele trata o processo como uma sequência de passos que devem ser cumpridos para que o próximo passo possa ser realizado. A cada passo, o botão correspondente é ativado, junto com uma mensagem de texto explicando o que será realizado naquela etapa.

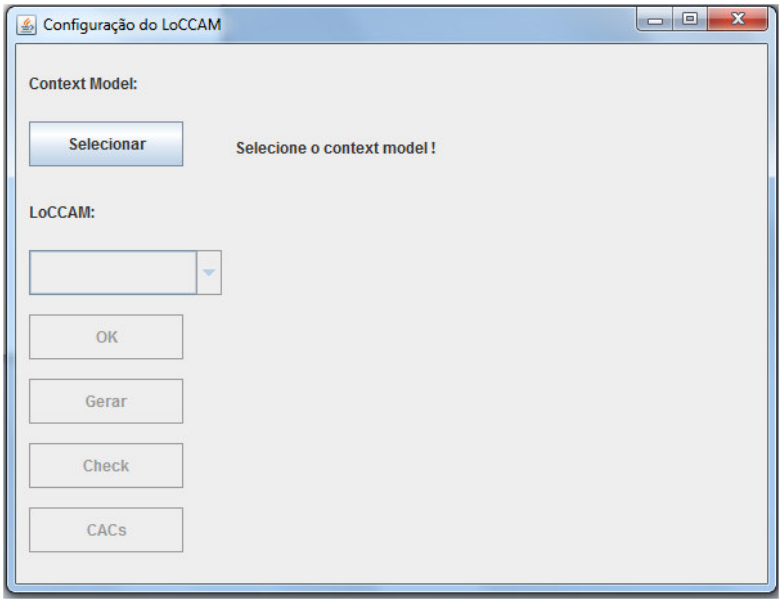


**Figura 4.16 – Interface do Instalador**

As duas primeiras etapas são a da escolha do modelo que deverá ser utilizado no processo e a seleção do dispositivo móvel no qual o LoCCAM e os CACs devem ser instalados. No instante em que o dispositivo é confirmado, é iniciado o processo de instalação do LoCCAM no dispositivo. Caso o *middleware* já esteja presente no aparelho, ele aborta o processo de instalação. O passo seguinte é a geração da lista de CACs. Uma vez que a lista esteja gerada, o passo seguinte é checar todas as *Context Keys* contidas na lista e a versão do Android presente no dispositivo. Com tudo pronto, o passo final é o desenvolvedor indicar em qual diretório estaria localizado o repositório local de CACs dele. O Instalador vasculha os CACs do diretório indicado procurando os CACs que contenham as *Context Keys*

correspondentes e que sejam de versão do Android compatível com a do dispositivo. Em caso de não conseguir encontrar algum dos CACs no diretório, o Instalador passa a procurar no Repositório *On-Line*. Ao final do processo de busca e instalação, o Instalador envia uma mensagem informando o término do processo.

## 4.7 Conclusões

Este capítulo apresentou a proposta da CRITiCAL, uma ferramenta de modelagem de informações e regras contextuais visando o desenvolvimento de aplicações móveis e sensíveis ao contexto que utilizem o LoCCAM como plataforma de *middleware* para aquisição de informações contextuais.

Foi apresentado uma visão geral do processo que envolve a CRITiCAL, posicionando-a em relação aos componentes que fazem parte de sua arquitetura interna, às transformações que a ferramenta realiza e os produtos que estas transformações geram. Além disso, foram explicados os princípios de *design* que nortearam o desenvolvimento desta ferramenta.

Foi apresentado o ContextRuleML, o metamodelo que possibilita a modelagem das informações contextuais e das regras. O ContextRuleML e a CRITiCAL foram posicionadas em relação às características apresentados na Seção 3.2 e chegou-se à conclusão de que eles satisfazem à maior parte das características levantadas. Além disso, foram apresentadas as evoluções que foram feitas no *middleware* LoCCAM durante o tempo de desenvolvimento da ferramenta.

Foi descrito o processo de geração do código, descrevendo como ele ocorria e apresentando os *templates* das regras de transformação. Além disso, foi descrito o produto final da transformação, um projeto Android com todos os métodos de comunicação entre LoCCAM e aplicação gerados, e esse produto foi detalhado.

Tanto foi descrito o processo de configuração do *middleware* em um dispositivo móvel a partir do modelo gerado pela ferramenta. A partir do modelo, é gerada uma lista de CACs, que é utilizada para fazer uma triagem dos CACs tanto em repositórios locais quanto no repositório online. A ferramenta instala o LoCCAM no dispositivo móvel, caso este ainda não esteja instalado, e, então, instala os CACs correspondentes e que sejam compatíveis com a versão do Android presente no dispositivo.

O próximo capítulo apresentará as avaliações realizadas como prova de conceito da ferramenta CRITiCAL. A primeira prova de conceito é uma avaliação de usabilidade, na qual

se buscou verificar a curva de aprendizado e a facilidade de uso e entendimento da ferramenta tanto na parte da modelagem das informações contextuais, quanto nas partes da geração de código e configuração de *middleware*. A segunda prova conceito é uma avaliação de desempenho que busca analisar se o código gerado pela CRITiCAL tem um desempenho igual ou similar ao código produzido manualmente utilizando os métodos de comunicação com o LoCCAM.

# Capítulo 5

# Avaliação

Neste capítulo são apresentadas as duas avaliações realizadas para analisar se os resultados alcançados por este trabalho correspondem aos objetivos esperados. Para a avaliação da CRITiCAL foi decidida a realização de duas avaliações distintas e independentes. A razão para tal foi poder validar, com melhor exatidão, as características da ferramenta:

- Ser uma ferramenta fácil de utilizar e de se modelar, mesmo para desenvolvedores que não tenham conhecimentos aprofundados acerca do funcionamento e da arquitetura interna do LoCCAM; e
- Prover um código gerado que não apresente grandes perdas de performace quando comparado a um código-fonte implementado diretamente pelo desenvolvedor.

Na Seção 5.1, é descrita a avaliação de usabilidade, detalhando o objetivo desta e todo o processo do planejamento aos resultados. Além de uma discussão sobre os resultados obtidos. Na Seção 5.2, é descrita a avaliação de desempenho. São detalhados os experimentos, o material utilizado nesses e os resultados obtidos, além de uma discussão acerca desses resultados.

## 5.1 Avaliação de Usabilidade

### 5.1.1 Objetivo da Avaliação

O objetivo desta avaliação é verificar a usabilidade da CRITiCAL tanto para desenvolvedores em geral, quanto para desenvolvedores com experiência prévia em implementações voltadas para o LoCCAM.

Com este fim, escolheu-se, de forma não probabilística e por conveniência, um grupo de 14 voluntários, todos da área da computação. Alguns dos voluntários possuiam experiência prévia no desenvolvimento com o LoCCAM. Para os 14 participantes lhes foi solicitado realizar uma série de atividades com intuito de se coletar dados sobre a usabilidade da

ferramenta e inferir a sua facilidade de uso e de aprendizado.

#### 5.1.1.1 Questões de Pesquisa

Para esta avaliação preliminar, foi necessário criar questões de pesquisa para se focar no que se pretendia ser obtido pelo experimento. Para tanto, as seguintes questões de pesquisa forma levantadas no experimento.

**Q1:** O quão simples de utilizar é a ferramenta CRITiCAL para desenvolvedores em geral?

**Análise:** Esta questão busca investigar se a ferramenta CRITiCAL é de fácil utilização para o desenvolvedor, bem como analisar quais etapas são mais fáceis ou mais complexas: a modelagem, a geração do código ou a configuração do *middleware*.

**Métrica:** O tempo gasto pelos voluntários em cada etapa da parte do desenvolvimento utilizando a CRITiCAL. Além de um conjunto de perguntas realizadas com base na escala Likert [BARBOSA E SILVA, 2010], nas quais os voluntários respondiam acerca da usabilidade da CRITiCAL.

**Q2:** A ferramenta CRITiCAL é mais simples de utilizar em comparação com a implementação utilizando diretamente o LoCCAM?

**Análise:** Esta questão busca analisar se, na opinião de desenvolvedores que já utilizaram o LoCCAM, a CRITiCAL é uma ferramenta que facilita o desenvolvimento das aplicações e em que etapas do desenvolvimento, eles consideram que há esse ganho em simplicidade.

**Métrica:** Um conjunto de perguntas realizadas, com base na escala Likert [BARBOSA E SILVA, 2010], nas quais os voluntários respondiam acerca da comparação entre as funcionalidades da CRITiCAL e o desenvolvimento utilizando apenas o LoCCAM.

**Q3:** Os modelos criados pela CRITiCAL são de fácil entendimento para desenvolvedores em geral?

**Análise:** Esta questão busca verificar se um desenvolvedor consegue compreender a semântica das regras contextuais construídas pela CRITiCAL com base em um modelo previamente feito por outro desenvolvedor.

**Métrica:** A quantidade de erros e acertos dos voluntários ao descreverem modelos previamente criados através da CRITiCAL.

### 5.1.1.2 Design da Avaliação Preliminar

A primeira etapa da avaliação consistiu em obter um grupo de voluntários, da área da computação, que pudessem realizar o experimento. Buscou-se encontrar um grupo variado de voluntários, priorizando-se um perfil de profissionais já graduados em cursos da área da computação e que, preferencialmente, tivessem experiência prévia em desenvolvimento com o *middleware* LoCCAM. Uma vez que esse grupo foi reunido, cada voluntário realizou as atividades descritas na Figura 5.1.

Inicialmente, todos os voluntários preencheram um questionário pré-experimento, com o objetivo de analisar o perfil deles e o seu grau de experiência nas ferramentas e abordagens utilizadas pela CRITiCAL.

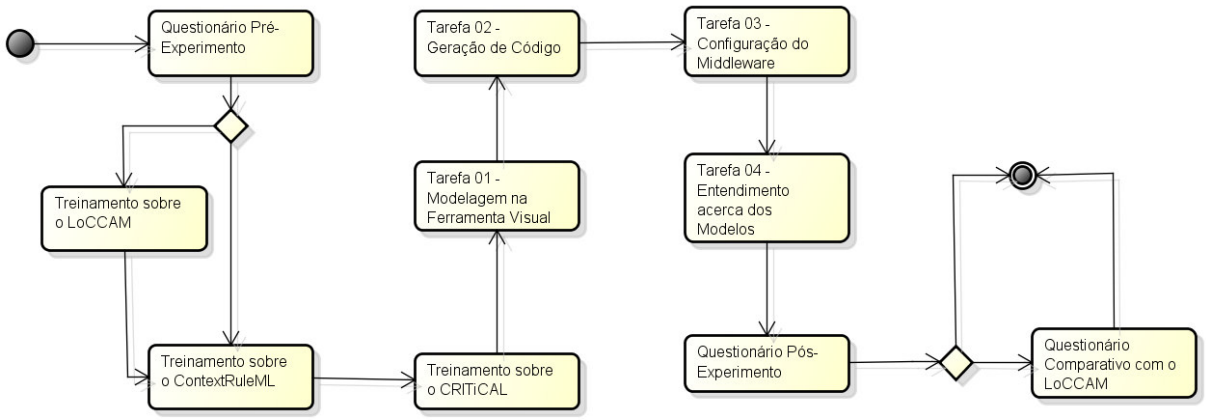


**Figura 5.1 – Atividades Realizadas na Avaliação Preliminar de Usabilidade**

Após isso, foi realizada a etapa de treinamento nos conceitos utilizados na CRITiCAL. Quatro dos voluntários tinham experiência prévia no desenvolvimento de aplicações utilizando o LoCCAM e não precisaram assistir o treinamento sobre o *middleware*, enquanto os demais passaram por esse treinamento. Foram realizados mais dois treinamentos. O primeiro foi focado na explicação da DSL ContextRuleML e sua sintaxe. O segundo apresentou a CRITiCAL, seu funcionamento e como ocorre a geração de código e configuração do *middleware*. Os voluntários também assistiram a um vídeo que mostrava a ferramenta sendo utilizada na prática.

Após os treinamentos, foram realizadas quatro tarefas utilizando a ferramenta. Foi descrita uma aplicação que exigia uma regra contextual com composições condicionais. Com base na descrição, a Tarefa 1 consistia no voluntário conseguir modelar essa descrição na ferramenta. Uma vez que o modelo estivesse pronto e salvo, a Tarefa 2 tratava da geração do código desse modelo e da importação do projeto Android gerado para a IDE Eclipse. A Tarefa

2 só era considerada completa quando o voluntário gerava um código que não apresentasse erros de compilação (o que configuraria um erro no modelo construído na Tarefa 1). A Tarefa 3 era composta pela configuração do *middleware* com base no modelo da Tarefa 1 e era considerada completa quando o voluntário conseguia instalar o LoCCAM e os componentes corretos no dispositivo móvel. A Tarefa 4 era uma tarefa visando o entendimento do voluntário a respeito da sintaxe dos modelos, com ele descrevendo dois modelos que lhes eram apresentados.

Ao final das Tarefas, cada voluntário respondia um questionário pós-experimento, com perguntas acerca do que ele achou da utilização da ferramenta CRITiCAL. Os voluntários com experiência prévia no LoCCAM também responderam um segundo questionário, mais focado na comparação entre o desenvolvimento para o LoCCAM utilizando a CRITiCAL e sem utilizar a CRITiCAL.

#### 5.1.1.3 Ameaças à Avaliação Preliminar

Entre as ameaças para a avaliação preliminar, pode-se citar a subjetividade das respostas dos voluntários.

Como uma mitigação para essa questão, foi calculado o tempo que os voluntários levaram para realizar cada uma das Tarefas. Assim, pode-se comparar os tempos gastos com as respostas das questões de usabilidade e analisar se as respostas estavam destoantes ou não do resultado prático.

### 5.1.2 Avaliação da Usabilidade

Com base no questionário pré-experimento respondido pelos voluntários, foi possível criar um perfil do nível de experiência prévia deles com as ferramentas e metodologias utilizadas na CRITiCAL. Foram realizadas um conjunto de sete perguntas, feitas na escala Likert [BARBOSA E SILVA, 2010], onde para cada pergunta, o usuário deve responder se "discordo totalmente", "discordo", "concordo", "concordo totalmente" ou "neutro" para uma declaração específica acerca de seu grau de experiência da ferramenta. O questionário é mostrado no Apêndice A. As respostas dos voluntários foram sumarizadas na Figura 5.2.

Também foi perguntado o grau de escolaridade dos voluntários. Eles incluíam estudantes de graduação (7%), doutorandos (21%), profissionais graduados (36%), mestrandos (36%) e mestres (7%). Todos os voluntários estudaram em cursos da área da computação, seja na Universidade Federal do Ceará (UFC) Campus Fortaleza e Campus

Quixadá, no Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) e Universidade de Fortaleza (UNIFOR).

A Figura 5.2 ilustra o perfil dos voluntários da avaliação. A maior parte dos voluntários reportou ter uma grande experiência na linguagem Java (78%) e na IDE Eclipse (64%). Uma significativa parte deles declarou ter um conhecimento do razoável ao grande no desenvolvimento de aplicações para Android (65%) e no desenvolvimento de aplicações móveis e sensíveis ao contexto (50%). Eles também reportaram ter pouca experiência no desenvolvimento de aplicações utilizando acesso direto a sensores (57%) e no desenvolvimento de aplicações utilizando o LoCCAM (79%), além de um baixo conhecimento a respeito dos conceitos da MDE (64%).

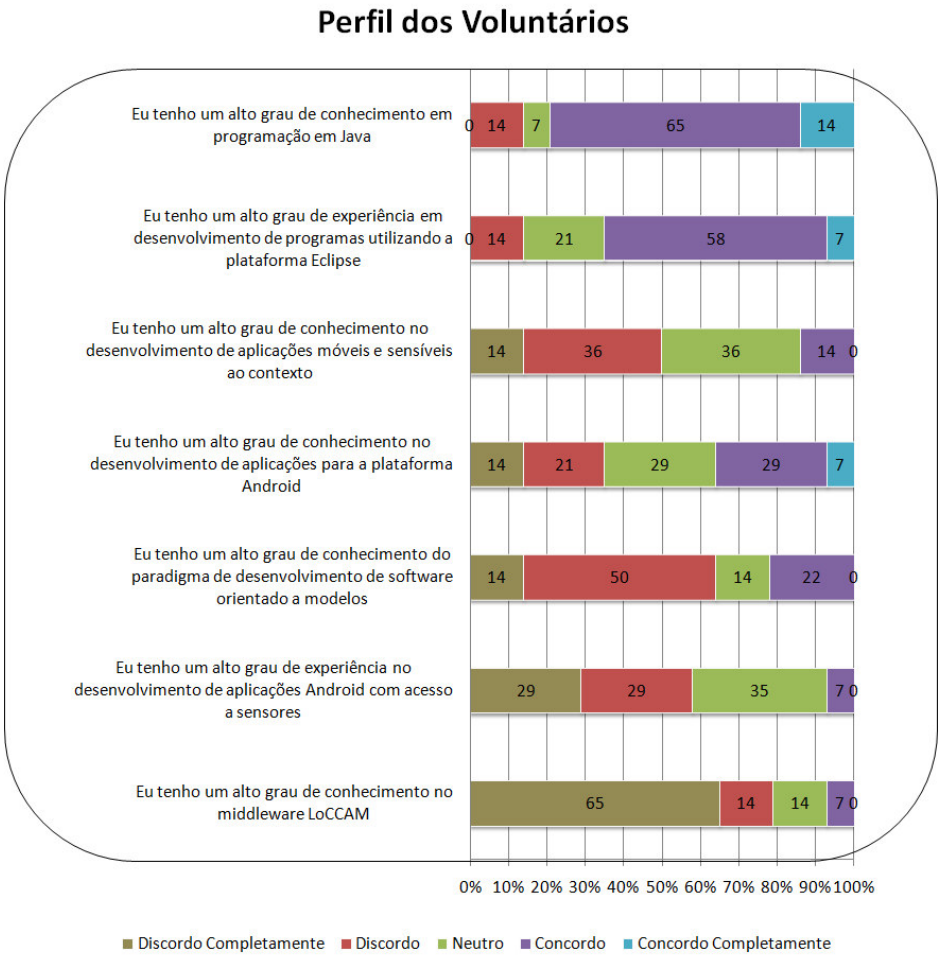


**Figura 5.2 – Perfil dos Voluntários**

Durante o treinamento e a realização das Tarefas especificadas na Subseção anterior, foi coletado o tempo que cada voluntário levou para realizar cada uma das etapas. Os tempos de execução de cada voluntário para o treinamento e cada uma das tarefas são mostrados na Tabela 5.1. O tempo médio total de treinamento dos voluntários foi de 26 minutos, sendo que

o tempo médio dos que conheciam previamente o LoCCAM foi de 20 minutos, enquanto o tempo médio dos voluntários que tiveram as três etapas do treinamento foi de, aproximadamente, 29 minutos. Isso demonstra que uma parte significativa do treinamento correspondia a uma explanação sobre os conceitos e métodos do *middleware*.

Uma vez realizada o treinamento foi calculado o tempo que os voluntários levavam para construir um modelo com uma regra contextual baseado na descrição abaixo.

"*Desenvolver um modelo que realize a ação "notifyApp" com as seguintes condições:*

- *Enquanto a aceleração do eixo Y for maior que 0 m/s² E*
- *Enquanto um número randômico gerado for maior que "10".* "

**Tabela 5.1 – Tabela com os Tempos de Execução de Cada Tarefa pelos Voluntários**

| Voluntário | Tempo Treinamento (min) | Tempo Tarefa 1 (min) | Tempo Tarefa 2 (min) | Tempo Tarefa 3 (min) |
|---|---|---|---|---|
| Voluntário 01 | 12 | 17 | 52 | 3 |
| Voluntário 02 | 12 | 20 | 47 | 4 |
| Voluntário 03 | 27 | 20 | 7 | 4 |
| Voluntário 04 | 27 | 28 | 4 | 3 |
| Voluntário 05 | 16 | 11 | 3 | 3 |
| Voluntário 06 | 30 | 14 | 3 | 2 |
| Voluntário 07 | 32 | 11 | 3 | 2 |
| Voluntário 08 | 36 | 13 | 4 | 3 |
| Voluntário 09 | 40 | 9 | 2 | 2 |
| Voluntário 10 | 26 | 11 | 21 | 4 |
| Voluntário 11 | 15 | 13 | 2 | 2 |
| Voluntário 12 | 36 | 16 | 3 | 2 |
| Voluntário 13 | 36 | 31 | 8 | 7 |
| Voluntário 14 | 23 | 14 | 2 | 3 |

Os voluntários levaram uma média de, aproximadamente, 16 minutos para realizarem a construção do modelo. Curiosamente, alguns dos que obtiveram os melhores tempos de desempenho nessa Tarefa estavam entre os que despenderam mais tempo de treinamento

(Voluntários 07 e 09). A Tarefa 01 era considerada concluída quando o voluntário autodeclarava que havia construído o modelo.

O tempo médio da Tarefa 02, correspondente à geração do código e importação do projeto gerado, foi de 11 minutos e 30 segundos. No entanto, em casos de código gerado apresentando erros de compilação, também era contabilizado nesta etapa o tempo que o voluntário levava para descobrir em que parte do modelo ele havia errado, corrigir tal falha e realizar uma nova geração de código. Os Voluntários 01, 02 e 10 apresentaram grandes dificuldades em entender em que parte do modelo estavam cometendo falhas. Desta forma, seus resultados foram pontos destoantes em comparação com a média geral dos voluntários. A média dos demais voluntários, excluindo estes três casos específicos, foi de aproximadamente 4 minutos.

O tempo médio gasto na Tarefa 03 foi de, aproximadamente, 3 minutos. Como os modelos já haviam sido construídos na Tarefa 01 e os eventuais erros corrigidos na Tarefa 02 era de se esperar um tempo mais baixo para esta etapa. Isso se deve ao fato de os voluntários já estarem mais experientes no uso da ferramenta e a execução desta Tarefa era bastante similar à Tarefa 02.

A Tarefa 04 consistia no voluntário descrever, com suas próprias palavras, os dois modelos apresentados na Figura 5.3. Ambos os modelos apresentavam apenas uma regra contextual. No entanto, estas eram compostas por várias condições, com cláusulas e informações contextuais diferentes.

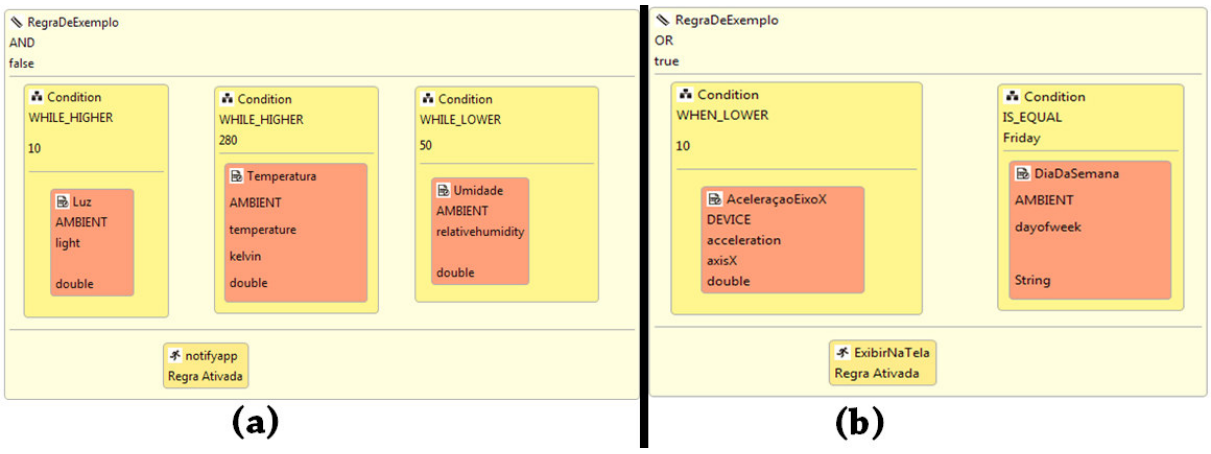


**Figura 5.3– Regras Utilizadas na Tarefa 04. A Regra 01 está expressa em (a) e a Regra 02 em (b)**

A Tabela 5.2 sumarizou os índices de erros e acertos dos voluntários. Era considerado "acerto" quando o voluntário descrevia de maneira integralmente certa os elementos da regra. O "erro" é informado nos casos em que os voluntários erraram a interpretação de algum dos

elementos e “incompleto” quando eles não informavam algum dos elementos essenciais da regra, como a cláusula das composições das condições.

A Regra 01 apresentou um índice de 57% de acerto, 36% de descrições incompletas e 7% de erros. A principal incidência nas descrições incompletas era a falta da definição da cláusula condicional, se as condições eram um “AND” ou um “OR”. A Regra 02 apresentou um índice de 43% de erros, 36% de acerto e 21% de descrições incompletas. Todas as descrições incompletas não apresentavam da definição da cláusula condicional, enquanto a maior parte dos erros era devido à imprecisão dos voluntários em distinguir a diferença entre regras de estado e regras de evento. Com base nisso, objetiva-se futuramente, tentar deixar, as diferenças entre Estado e Evento mais didáticas nos modelos. As descrições completas das respostas dos voluntários são mostradas no Apêndice B.

**Tabela 5.2 – Resultado Acerto/Erro dos Voluntários em Relação à Tarefa 04 do Experimento**

| Voluntário | Regra 01 | Regra 02 |
|---|---|---|
| Voluntário 01 | Acerto | Acerto |
| Voluntário 02 | Acerto | Erro<br>(confusão entre estado e evento) |
| Voluntário 03 | Erro<br>(erro da categoria contextual) | Erro<br>(confusão entre estado e evento) |
| Voluntário 04 | Acerto | Erro<br>(confusão entre estado e evento) |
| Voluntário 05 | Acerto | Erro<br>(confusão entre estado e evento) |
| Voluntário 06 | Acerto | Erro<br>(confusão entre “AND” e “OR”) |
| Voluntário 07 | Incompleto<br>(não definiu um dos valores) | Acerto |
| Voluntário 08 | Acerto | Acerto |
| Voluntário 09 | Incompleto<br>(não definiu cláusula condicional) | Incompleto<br>(não definiu cláusula condicional) |
| Voluntário 10 | Incompleto<br>(não definiu cláusula condicional) | Incompleto<br>(não definiu cláusula condicional) |
| Voluntário 11 | Acerto | Incompleto<br>(não definiu cláusula condicional) |
| Voluntário 12 | Incompleto<br>(não definiu cláusula condicional) | Acerto |
| Voluntário 13 | Incompleto<br>(não definiu um dos valores) | Acerto |
| Voluntário 14 | Acerto | Erro<br>(confusão entre “AND” e “OR”) |

Após a realização das Tarefas, os voluntários responderam a um questionário pós-experimento, com perguntas na escala Likert [BARBOSA E SILVA, 2010] a respeito das características da CRITiCAL. O conjunto de oito perguntas seguia a estrutura do questionário

pré-experimento e é mostrado no Apêndice C. As respostas dos voluntários foram sumarizadas na Figura 5.4.

No geral, o *feedback* dos voluntários foi positivo e eles apresentaram uma boa aceitação da ferramenta. Dentre as questões, foi unânime entre os usuários de que a CRITiCAL é uma ferramenta fácil de ser utilizada (100%), sendo que 84% dos voluntários concordaram que a utilizariam com freqüência ao desenvolver aplicações para o LoCCAM. A maior parte dos voluntários afirmou que a ferramenta era consistente (64%), considerou todas as funcionalidades bem integradas (50%) e se sentiu confiante em utilizar a ferramenta (50%). A maior parte dos voluntários considerou a CRITiCAL uma ferramenta de rápido aprendizado (78%) e que não precisaram adquirir muitos conhecimentos novos para se tornarem aptos a utilizá-la (72%). No entanto, apenas 14% concordaram que a utilização da ferramenta não necessita de um treinamento prévio.

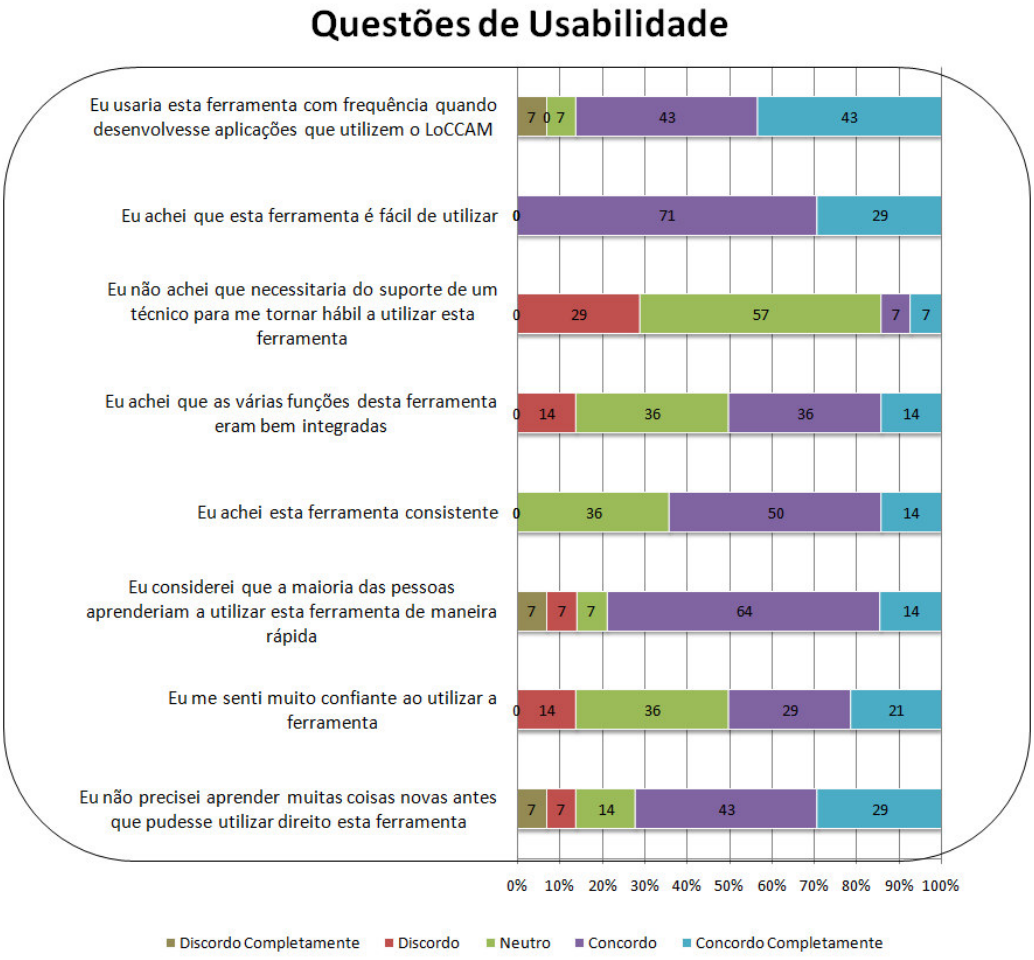


**Figura 5.4 – Resultado das Questões de Usabilidade**

Com base nisso, pode-se chegar à conclusão de que a CRITiCAL é uma ferramenta de fácil utilização, mas que é importante aperfeiçoar o processo de aprendizado de sua utilização.

Além da disponibilização de tutoriais e vídeos-tutoriais sobre como utilizar a ferramenta, também seria útil um aperfeiçoamento da interface gráfica para torná-la mais intuitiva.

### 5.1.3 Avaliação Comparativa com o LoCCAM

No caso dos voluntários com experiência prévia no desenvolvimento de aplicações utilizando o LoCCAM, também foi pedido que eles respondessem a um segundo questionário pós-experimento, com perguntas na escala Likert [BARBOSA E SILVA, 2010] a respeito da comparação entre o desenvolvimento utilizando a CRITiCAL em contraposição ao desenvolvimento implementando diretamente com o LoCCAM. O conjunto de cinco perguntas seguia a estrutura dos questionários de pré e pós-experimento e é mostrado no Apêndice D. As respostas dos voluntários foram sumarizadas na Figura 5.5.

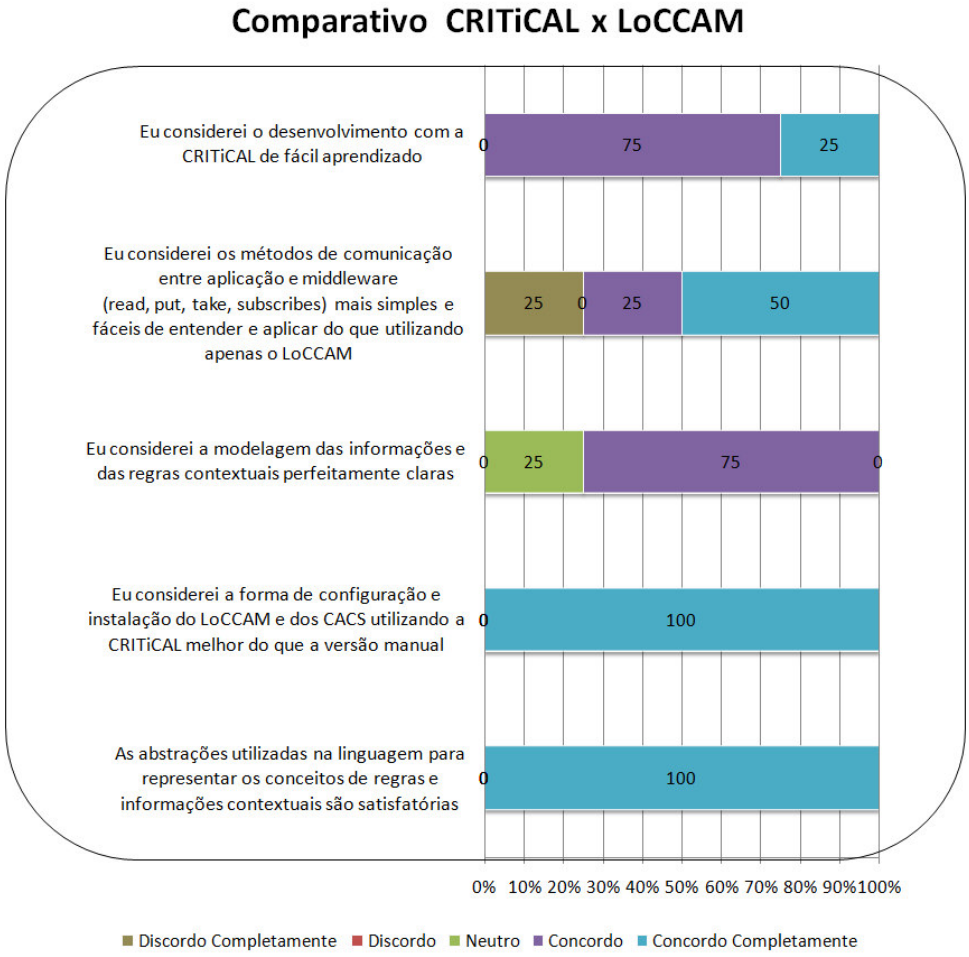


**Figura 5.5– Resultado do Questionário Comparativo entre o desenvolvimento com a CRITiCAL e utilizando o LoCCAM manualmente**

No geral, as respostas dos voluntários foram quase unanimemente positivas em considerar o desenvolvimento utilizando com a CRITiCAL mais prático e simples do que a implementação simples e direta com LoCCAM. Todos os voluntários concordaram que o

desenvolvimento com a CRITiCAL era de fácil aprendizado (100%) e a maioria deles concordou que os métodos de comunicação entre aplicação e *middleware* eram mais fáceis de entender e de aplicar do que implementando diretamente pelo LoCCAM (75%). A maioria dos voluntários considerou a modelagem das informações contextuais e regras contextuais perfeitamente claras e entendíveis. Todos os voluntários concordam completamente que a configuração e instalação do LoCCAM e dos CACs através da CRITiCAL era melhor que o método manual (100%) e todos também concordaram completamente que as abstrações utilizadas na ferramenta para representar os conceitos das informações e regras contextuais eram satisfatórias (100%).

Com base em um *feedback* tão positivo de desenvolvedores com experiência prévia com o LoCCAM, pode-se chegar à conclusão de que o desenvolvimento com a CRITiCAL é significativamente mais simples do que comparado à implementação dos métodos manualmente no domínio dos desenvolvedores com experiência média com o LoCCAM.

### 5.1.4 Problemas Identificados

No questionário pós-experimento, existia um campo para que os voluntários pudessem expressar de maneira mais subjetiva o seu *feedback*. Assim, eles poderiam sugerir melhorias ou mencionar lacunas e problemas que eles observaram na ferramenta. Os comentários dos voluntários no questionário pós-experimento são apresentados na Tabela 5.3, separando os comentários dos problemas que eles observaram.

Um dos principais problemas mencionados pelos voluntários foi a ausência de uma validação do modelo antes da geração do código, o que resultava na geração de código com erros de compilação. Além disso, muitos apresentaram sugestões para maior simplificação mais do uso da ferramenta e torná-la mais visualmente intuitiva. Outros fizeram sugestões para a criação de novos CACs. Todos os *feedbacks* foram guardados para servir como guia para o aperfeiçoamento da ferramenta em trabalhos futuros.

**Tabela 5.3 – Comentários e Problemas Observados pelos voluntários no Experimento de Usabilidade**

| Comentários dos Voluntários | Problemas Observados |
|---|---|
| • "*Muito boa a ferramenta.*"<br>• "*A ferramenta é simples e não necessitou conhecer a estrutura do LoCCAM pra poder utiliza-la.*"<br>• "*Parabéns.*"<br>• "*A estrutura visual dos blocos está muito boa.*"<br>• "*Ferramenta com baixa curva de aprendizado. Feedeback satisfatório da ferramenta durante o uso.*"<br>• "*Gostei bastante da ideia da ferramenta.*"<br>• "*A manipulação da ferramenta é bastante fácil, porém tem um excesso de informação que, inicialmente, pode acabar por confundir o usuário. Mas basta um acompanhamento rápido e o problema é sanado.*" | • "*Mostrar um exemplo da aplicação no final para o voluntário ver visualmente o resultado do que ele vez.*"<br>• "*Retirar dos slides do treinamento tudo que for desnecessário.*"<br>• "*Faça os botões funcionarem.*"<br>• "*O problema que ocorreu em minha atividade diminuiu minha credibilidade na ferramenta nesta versão inicial. Apesar de acreditar que a ferramenta, quando numa versão mais estável, será muito útil e de fácil utilização.*"<br>• "*Algumas informações que possuem poucas opções são indicadas em texto livre. Elas poderiam ser informadas através de um combo (ENUM).*"<br>• "*Acho que algumas operações podem ser encapsuladas na forma de botões, isso irá facilitar ainda mais o uso da ferramenta.*"<br>• "*Senti falta da validação dos parâmetros inseridos. Coloquei nomes iguais e o código gerado teve erros.*"<br>• "*Tratar exceção, limitar números random e integrar todas as ferramentas em uma só.*" |

Os voluntários com experiência prévia no desenvolvimento com o LoCCAM também tiveram que tecer comentários e falar sobre problemas encontrados em relação à comparação com a implementação manual. Esses comentários são mostrados na Tabela 5.4.

**Tabela 5.4 – Comentários e Problemas Observados pelos voluntários experientes no desenvolvimento com o LoCCAM**

| Comentários dos Voluntários | Problemas Observados dos |
|---|---|
| • "*Parabéns.*"<br>• "*Gostei muito da ferramenta. Facilita muito a utilização do middleware LoCCAM e dos CACs. Abstrai bastante coisa e poupa muito tempo do desenvolvedor.*"<br>• "*A ferramenta Loccam requer muito tempo de aprendizado, é complexa para pequenas tarefas e requer um trabalho maior que utilizando bibliotecas nativas. Ainda falta algo mais prático na utilização do Loccam, ferramentas como o CRITiCAL podem ajudar na expansão do Loccam. Parabéns!*" | • "*Na criação dos modelos, achei que não ficou tão claro onde cada coisa deveria estar. Nesse ponto, senti falta de algo mais intuitivo, ou com alguma legenda, algo que indicasse o que e onde inserir os 'componentes'. Porém depois de receber instruções, o processo fica simples.*"<br>• *"O LoCCAM deveria ser invisível e transparente ao usuário.*" |

## 5.2 Avaliação de Desempenho

Além da avaliação de usabilidade, também foi realizada uma avaliação do desempenho das aplicações construídas com base no código gerado pela CRITiCAL. O objetivo desta avaliação era comparar o desempenho de uma mesma aplicação com o código gerado pela CRITiCAL e sem a utilização deste, a fim de analisar possível decrescimentos na performance devido à utilização da ferramenta.

### 5.2.1 Implementação da LightLoCCAM

Com o objetivo de realizar a avaliação de desempenho, foi desenvolvida uma aplicação simples, a LightLoCCAM.

A LightLoCCAM é composta de uma única regra. A aplicação realiza a leitura contínua da luminosidade. E enquanto o valor da luminosidade for menor que 5 lux, a cor da lâmpada muda de branco para verde, indicando a variação. A mudança no comportamento é mostrada pela Figure 5.6.

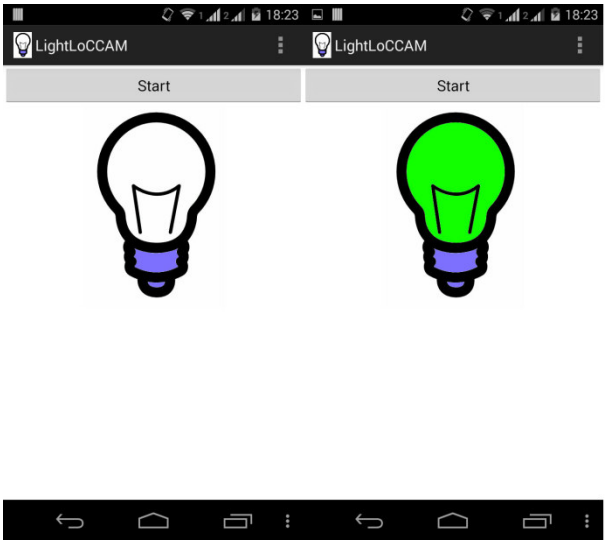


**Figura 5.6– LightLoCCAM**

Foi decidido pelo desenvolvimento de uma aplicação simples para a avaliação do desempenho, pela facilidade da replicação da LightLoCCAM nas três abordagens distintas (usando a CRITiCAL, sem o uso da CRITiCAL e o sem a CRITiCAL e o LoCCAM) e pela simplicidade em gerar simulações de variação no ambiente.

### 5.2.2 Descrição do Experimento

O experimento foi realizado em duas etapas. A primeira etapa consistia em realizar a avaliação da memória livre nos três casos, enquanto a segunda parte mensurava o tempo de resposta de cada abordagem. Ambas as etapas foram em separado, pelo mesmo desenvolvedor, e cada qual com uma estratégia de coleta diferente. No entanto, elas tinham em comum o fato de terem sido desenvolvidas três versões da LightLoCCAM. A primeira desenvolvida utilizando os métodos tradicionais de comunicação com o LoCCAM. A segunda com base no código gerado pela CRITiCAL. A terceira implementada com os métodos do Android para acesso direto aos sensores. Desta maneira, objetivou-se uma comparação equiparada entre as três abordagens para o cálculo do desempenho.

No experimento da memória livre disponível, foram executadas quatro cópias da LightLoCCAM simultaneamente para cada uma das três abordagens. O tempo de execução do experimento de cada abordagem foi de 2 minutos. A cada 1 segundo, era realizado a coleta da memória livre disponível no dispositivo móvel em KB. Apenas uma das cópias da aplicação realizava a coleta dos dados da memória em abordagem. O experimento foi repetido 10 vezes para cada uma das três abordagens. Os dados coletados eram salvos em um arquivo de texto para a posterior análise.

No experimento do tempo de resposta, foi executada apenas uma cópia da LightLoCCAM para cada uma das três abordagens. Foi realizada uma adaptação na LightLoCCAM para que esta pudesse realizar continuamente a leitura do valor da luminosidade ambiente. Assim, foi calculado a soma do tempo que cada aplicação demorava em iniciar a comunicação com o LoCCAM, publicar o interessar o interessa da informação contextual e realizar uma leitura no espaço de tuplas. Na versão sem a participação do *middleware*, foi calculado apenas o tempo do acesso do sensor e da resposta da informação. O tempo de resposta foi calculado 120 vezes. Os dados coletados também foram salvos em um arquivo de texto para a posterior análise.

### 5.2.3 Material Utilizado

Em ambas as etapas do experimento de avaliação, foram utilizadas um *smartphone* Motorola com Android versão 4.4.4. Suas configurações eram CPU com processador Qualcomm Snapdragon 400 MSM8226 / ARM Cortex-A7Clock, 8GB de memória interna e 1,0 GB de memória RAM.

### 5.2.4 Resultados

Uma vez que os dados foram coletados e armazenados nos arquivos de texto, se partiu para as etapas de análise dos dados brutos e discussão das inferências obtidas através destes. No caso da avaliação da memória livre a partir dos dados das 10 iterações feitas no experimento, foi realizada uma média simples com o valor médio da memória livre de cada segundo em KB. Esses valores foram convertidos em MB. Assim, obteve-se, para cada abordagem, uma tabela com a memória livre média durante a execução do experimento em MB do segundo 01 ao segundo 120.

As três tabelas foram, então, comparadas em um gráfico de Memória Livre (em MB) pelo Tempo (em segundos). Esse gráfico é mostrado na Figura 5.7. O gráfico foi normalizado de modo a todas as abordagens inicializarem com a mesma memória livre. É perceptível que as abordagens com o CRITiCAL e somente o LoCCAM apresentam um comportamento bastante similar, estabilizando-se na mesma faixa de desempenho ao longo do tempo. A diferença entre o desempenho destes com o acesso livre dos sensores deve-se à utilização do *middleware*, fazendo com que a memória livre de ambas fosse menor por ele estar ativo. Como a LightLoCCAM é uma aplicação simples, que requer o uso de uma única informação contextual e não requer a realização de nenhuma adaptação dinâmica, era esperado que, neste

caso, o acesso direto tivesse um desempenho melhor. Assim, o objetivo era a comparação entre as abordagens com *middleware* utilizando a CRITiCAL e sem a CRITiCAL, a fim de verificar uma provável perda de desempenho. No entanto, foi verificado que o desempenho da aplicação permanece.

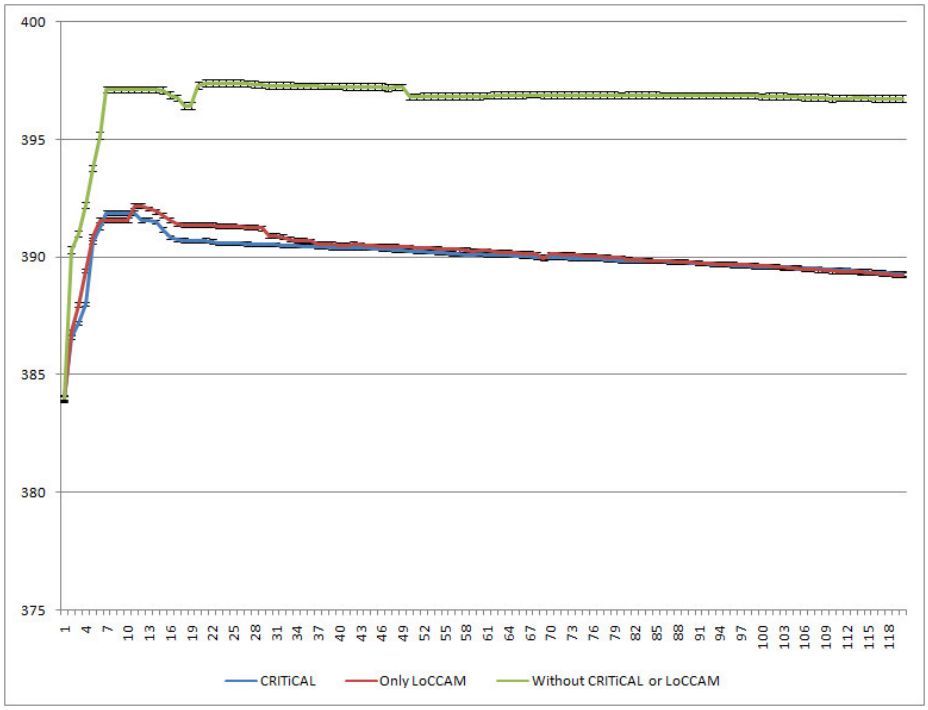


**Figura 5.7– Gráfico Comparativo da Memória Livre – Memória Livre (MB) x Tempo (s)**

No caso da avaliação de tempo de respostas, os dados brutos coletados e armazenados nos arquivos de texto estavam na escala de nanosegundos (ns). A primeira etapa da análise consistiu em converter esses valores em milissegundos (ms) e calcular a média aritmética dos 120 valores e a sua variância. No entanto, um dos valores, correspondente à 32ª iteração da leitura do valor para a abordagem sem a CRITiCAL e com o LoCCAM, mostrou-se bastante díspare, quase o triplo da média (80,892 ms). Desta maneira, foi decidido pela retirada deste valor para o cálculo da média e variância dessa abordagem, bem como das 32ªs interações do cálculo das demais abordagens.

A tabela 5.5 apresenta os resultados das médias de tempo de resposta em milissegundos e a variância obtida para as três abordagens. Já era esperada a diferença de tempo de resposta na abordagem sem *middleware*, uma vez que nas demais abordagens também foi calculado o tempo para a inicialização da comunicação com o *middleware* e publicação do interesse na informação contextual.

**Tabela 5.5 – Resultado do Experimento de Tempo de Resposta para as Três Versões do LightLoCCAM**

| Versão da Aplicação | Média do Tempo de Resposta (ms) | Variância |
|---|---|---|
| LightLoCCAM criado com o CRITiCAL | 34,532 ms | 8,877 |
| LightLoCCAM usando apenas LoCCAM | 35,432 ms | 15,480 |
| LightLoCCAM realizando acesso direto ao sensor | 5,473 ms | 0,085 |

Dentre as abordagens que utilizaram o LoCCAM, a abordagem com a CRITiCAL teve um desempenho de tempo de resposta levemente melhor. No entanto, a sua variância foi significativamente menor que a versão sem a CRITiCAL. Com isso, pode-se chegar à conclusão de que a versão utilizando a CRITiCAL tem um desempenho mais estável, realizando a leitura da informação contextual em um intervalo de tempo menor e mais previsível que a versão construída sem a utilização da ferramenta.

Outra métrica analisada correlaciona o número de linhas de código que as três versões da LightLoCCAM, fazendo uma distinção entre o número de linhas de código total presentes nas classes da aplicação e o número de linhas de código efetivamente escritas pelo desenvolvedor. Esse número de linhas efetivamente escritas exclui as linhas correspondente aos trechos de código gerados automaticamente pela IDE Eclipse ou pela CRITiCAL. A Tabela 5.6 apresenta o resultado desta análise.

**Tabela 5.6 – Número de Linhas de Código Efetivamente Escritas pelo Desenvolvedor para as Três Versões do LightLoCCAM**

| Versão da Aplicação | Número de Linhas de Código Total na Aplicação | Número de Linhas de Código Escritas pelo Desenvolvedor |
|---|---|---|
| LightLoCCAM criado com o CRITiCAL | 386 linhas | 71 linhas |
| LightLoCCAM usando apenas LoCCAM | 198 linhas | 164 linhas |
| LightLoCCAM realizando acesso direto ao sensor | 146 linhas | 112 linhas |

Analisando o resultado apresentado na Tabela 5.6, pode-se inferir que o número de linhas de código efetivamente escritas pelo desenvolvedor diminui sensivelmente através da ferramenta CRITiCAL. A maior parte das 71 linhas escritas pelo desenvolvedor na versão da

LightLoCCAM criada pela CRITiCAL foram na implementação do botão presente da *Activity* e no desenvolvimento dos métodos de coleta de memória livre. No que se refere à chamada da regra contextual, o desenvolvedor precisou escrever apenas 02 linhas de código. Outro detalhe a ser observado é que a CRITiCAL gera automaticamente a documentação dos métodos, aumentando o número de linhas de código total, enquanto as outras duas versões não foram documentadas, nem mesmo manualmente. Foram contabilizadas 67 linhas de comentários gerados pela CRITiCAL dentre as 386 linhas presentes nas duas classes da aplicação LightLoCCAM.

Dessa forma, pode-se chegar à conclusão de que o desenvolvimento utilizando a CRITiCAL é capaz de diminuir a complexidade e o tempo de desenvolvimento quando comparada ao desenvolvimento utilizando apenas o LoCCAM ou sem a utilização de nenhuma plataforma de *middleware*.

Com base nos dados obtidos pelos dois experimentos e pela métrica da linha de código, pode-se chegar à conclusão de que as aplicações construídas utilizando a CRITiCAL tem um desempenho similar às aplicações construídas com implementação tradicional dos métodos do LoCCAM. Assim, pode-se comprovar que, através da CRITiCAL, é possível gerar um código para uma aplicação móvel sensível ao contexto sem perda de desempenho.

## 5.4 Conclusão

Neste capítulo foram apresentadas as duas avaliações realizadas para validar as características da CRITiCAL.

Na avaliação de usabilidade, foram apresentadas as questões de pesquisa, os critérios de escolha dos voluntários para a avaliação e o fluxograma das atividades realizadas por eles nos experimentos. Então, os resultados foram sumarizados e cada etapa da avaliação foi detalhada e discutida a respeito das conclusões que se pode inferir dos dados. Chegou-se à conclusão de que a CRITiCAL é uma ferramenta de fácil utilização e aprendizado, mas cuja interface pode ser aperfeiçoada para tornar-se mais intuitiva e que sua utilização plena requer um treinamento prévio.

Na avaliação de desempenho, foi apresentada a aplicação LoCCAMLight, criada com base no código gerado pela ferramenta CRITiCAL. Com base nesta, foram criadas duas outras aplicações similares, uma que realizava acesso direto a sensores e a outra que implementava diretamente os métodos de comunicação com o LoCCAM, sem o uso da CRITiCAL ou da

LoCCAM_Lib. Com essas três aplicações, foram realizadas dois experimentos. O primeiro mensurava a memória livre do sistema e o segundo, o tempo de resposta da leitura de uma informação contextual. Os dados puros obtidos foram tratados, normalizados e comparados em gráficos. Assim, pode-se concluir que aplicações que utilizam o código gerado pela CRITiCAL não sofrem perda de desempenho quando comparadas com uma aplicação similar com implementação dos métodos do LoCCAM realizada de forma tradicional.

Por fim, com base nos resultados da avaliação de usabilidade e nos comentários relatados pelos voluntários, se pode encontrar algumas lacunas e questões da ferramenta que devem ser aperfeiçoadas futuramente.

# Capítulo 6

# Conclusão e Trabalhos Futuros

O objetivo deste capítulo é sumarizar os resultados obtidos e as conclusões acerca destes resultados. Além disso, também é apresentado um conjunto de possibilidades de trabalhos futuros que podem ser continuados a partir do estado atual dessa dissertação. Na Seção 6.1,é apresentado um resumo dos resultados alcançados neste trabalho. Na Seção 6.2, é apresentada a produção bibliográfica realizada durante o desenvolvimento deste trabalho. A Seção 6.3 discute as limitações ainda existentes na ferramenta CRITiCAL. E, por fim, a Seção 6.4 apresenta os trabalhos futuros que podem ser derivados do trabalho apresentado nesta dissertação.

## 6.1 Resultados Alcançados

Este trabalho apresenta a CRITiCAL, uma ferramenta que permite a modelagem visual de informações e regras contextuais e, com base nestas, a geração automatizada do código de comunicação entre uma aplicação Android e o *middleware* LoCCAM. A DSL proposta,ContextRuleML,possibilita a modelagem das informações contextuais e das regras contextuais.

A CRITiCAL fornece uma forte integração harmoniosa com o *middleware* LoCCAM e a DSL ContextRuleML. Em conjunto, auxiliam desenvolvedores a lidar com a complexidade do código inerente ao gerenciamento de contexto e ao acesso aos sensores do dispositivo.

A CRITiCAL apresenta dois processos de transformação com base em um modelo criado pela ferramenta: a geração do Projeto Android e a configuração do *middleware*.

No processo de geração do projeto Android, o modelo para a geração de duas classes que encapsulam todos os métodos de comunicação entre o *middleware* e a aplicação a ser desenvolvida. No processo de configuração do *middleware*, o modelo serve como base para a criação de uma lista de componentes que devem ser instalados de forma automatizada no dispositivo móvel junto com o LoCCAM. Caso o desenvolvedor não tenha esses componentes

em sua máquina, existe a possibilidade de realizar o download destes componentes a partir do Repositório *On-Line* de CACs.

Para validar a CRITiCAL, foram realizadas duas avaliações. A primeira avaliação, de usabilidade, visava analisar a curva de aprendizado e o nível de dificuldade na utilização da ferramenta. Para tanto, foram selecionas 14 voluntários, todos da área da computação, para realizar um conjunto de atividades. Com base nestas, pode-se inferir a CRITiCAL é uma ferramenta de rápido aprendizado e utilização simples. No entanto, foram observadas lacunas no design da interface e problemas a serem corrigidos na ferramenta.

A segunda avaliação, de desempenho, visava analisar a performance de uma aplicação criada com base no código gerado pela ferramenta CRITiCAL quando comparada com aplicações similares que não utilizaram a abordagem. Foram analisadas a memória consumida pelas aplicações (com base na memória livre disponível) e o tempo de resposta das aplicações. Através disso, pode-se inferir que as aplicações que utilizavam o código gerado pela CRITiCAL tem um desempenho similar ao das aplicações que utilizam o LoCCAM, mas sem usar código gerado pela ferramenta.

## 6.2 Produção Bibliográfica

Durante o período do Mestrado, foram publicados dois artigos focando em partes da ferramenta ou em estágios intermediários da abordagem. Foram eles:

- *Uma abordagem baseada em modelos para a geração de aplicações conscientes de contexto*. Publicado como short paper no XX Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), apresenta um estágio intermediário da abordagem, focando-se na DSL ContextRuleML e na visão geral do processo da CRITiCAL. Autores: Paulo Artur de Sousa Duarte, Felipe Mota Barreto, Francisco Anderson de Almada Gomes, Windson Viana de Carvalho e Fernando Antonio Mota Trinta
- *LoCCAMConfigurator: Uma Ferramenta para Modelagem Visual de Configurações de um Middleware de Suporte à Aplicações Conscientes de Contexto*. Publicado no XIII Workshop de Ferramentas e Aplicações (WFA), apresenta um trabalho focado na ferramenta (que ainda tinha o nome provisório de "LoCCAMConfigurator") e nas suas funcionalidades. Autores: Paulo Artur de Sousa Duarte, Francisco Anderson de Almada Gomes, Felipe Mota Barreto, Windson Viana de Carvalho e Fernando Antonio Mota Trinta.

- *Towards Context-Aware Behaviour Generation*. Publicado como short paper no 30th ACM/SIGAPP Symposium On Applied Computing (SAC), apresenta um estágio intermediário da abordagem, focando-se na visão geral do processo da CRITiCAL e nas transformações realizadas. Autores: Paulo Artur de Sousa Duarte, Francisco Anderson de Almada Gomes, Felipe Mota Barreto, Windson Viana de Carvalho e Fernando Antonio Mota Trinta.

## 6.3 Limitações

A CRITiCAL apresenta algumas limitações que puderam ser identificadas desde sua concepção. A principal delas é o seu forte acoplamento ao *middleware* LoCCAM, tornando-a uma ferramenta dependente de plataforma. Isso também acarreta no fato da CRITiCAL ter as mesmas limitações atuais do LoCCAM, uma vez que ela só gera código compatível com as plataformas de dispositivos móveis que sejam compatíveis com o *middleware*. Assim sendo, plataformas de desenvolvimento com importante participação no mercado, como o iOS e o Windows Phone, acabam por não serem contempladas com códigos compatíveis para estas.

Além disso, a CRITiCAL requer, para seu funcionamento pleno, um ambiente de desenvolvimento previamente configurado. Este ambiente seria uma IDE Eclipse com os *plugins* do GMF, xPand, MWE e o ADT *bundle* do Android instalados, além de ter o caminho do ADB (*Android Debug Bridge*)[19] da SDK Android devidamente adicionado no PATH das variáveis de ambiente do Sistema Operacional utilizado.

Outras limitações existentes na abordagem apresentada são inerentes às limitações existentes no paradigma da *model-driven engineering*. Uma das quais é a impossibilidade de manutenção diretamente no código gerado sem que este se afaste do modelo construído. Ou a atualização do modelo com posterior re-geração do código sem que isso acarrete na perda das modificações feitas diretamente no código.

## 6.4 Trabalhos Futuros

Entre os possíveis trabalhos futuros que poderão ser desenvolvidos para reduzir as limitações da ferramenta e ampliar o impacto da pesquisa realizada, pode-se citar:

- **Validação do modelo antes da geração do código:** um dos problemas observados durante a avaliação de usabilidade foi a ausência de uma validação dos modelos

19 http://developer.android.com/tools/help/adb.html

construídos antes da geração do código. Isso resultava na geração de projetos que apresentavam erros de compilação. Pretende-se evitar esse tipo de problema com a realização da validação do modelo. Assim, evitar-se-iam erros mais comuns no desenvolvimento dos modelos, tais como o esquecimento do preenchimento de alguns parâmetros ou componentes inteiros, além do uso de um mesmo nome para duas condições, regras ou informações contextuais.

- **Adição de Informações Contextuais e Condições na Paleta da Ferramenta:** Pretende-se realizar a inserção de ícones representando informações contextuais mais comuns (e.g., luminosidade, temperatura ambiental) e condições comuns envolvendo informações contextuais de localização (distância de um certo ponto, entrada/saída de uma determinada região).
- **Paleta dinâmica e adaptativa:**Uma vez que as informações contextuais e as condições mais comuns e utilizadas estejam na paleta de ícones da ferramenta, pretende-se que esta se torne dinâmica e adaptativa. Desta maneira, através de um banco de dados contendo a relação entre os dispositivos móveis e os sensores disponíveis neste, a paleta de ícones poderia adaptar-se para exibir apenas as informações contextuais que podem ser inferidas com bases nos sensores existentes no dispositivo selecionado, ou no grupo de dispositivos selecionados. Isso facilitaria o desenvolvimento de aplicações que sejam voltadas especificamente para determinadas famílias de dispositivos móveis.
- **Disponibilização dos CACs mais complexos:** A maior parte dos CACs presentes no repositório são *wrappers* de acesso simples a sensores. Pretende-se que sejam construídos mais CACs do tipo lógico, que recebam as informações contextuais de outros CACs e realizem inferências mais complexas a partir destes.
- **Repositório de CACs colaborativo:** Pretende-se que o repositório de CACs seja expandido e melhor organizado, de modo a torná-lo colaborativo. Assim, qualquer desenvolvedor que pretenda criar um CAC e disponibilizá-lo para a comunidade poderia o fazer no repositório oficial e este passar a ficar disponível para o *download* automático quando necessário por algum dispositivo.
- **Uso da CRITiCAL no ensino:**A partir de 2015, pretende-se aplicar a CRITiCAL em um curso de graduação focado no ensino de computação móvel e sensível ao contexto. Com isso, espera-se que a abordagem possa ajudar no processo de aprendizagem dos

principais conceitos de computação ubíqua e facilitar o desenvolvimento de aplicações sensíveis ao contexto pelos alunos.

- **Aperfeiçoamento dos *templates* de geração de código:** Pretende-se aperfeiçoar os *templates* de geração de código. Entre as melhorias pretendidas, pode-se mencionar a adição de mais ações pré-estabelecidas. O objetivo é fazer com os *templates* permitam a geração de uma nova classe, que encapsularia todas as ações que foram definidas no modelo.
- **Geração de aplicações inteiras a partir de modelos:**Por fim, objetiva-se a criação de uma ferramenta que permita a modelagem visual completa de uma aplicação móvel e sensível ao contexto. Essa ferramenta receberia um modelo construído pela CRITiCAL e, com base neste, criaria um modelo mais específico, no qual o desenvolvedor poderia modelar toda a lógica da aplicação. Ao final desta, seria gerado um projeto Android com código completo da aplicação.

# Referências Bibliográficas


ALMEIDA, J.P.A.; IACOB, M.E.; JONKERS, H.; QUARTEL, D.A.C. Model-driven development of context-aware services, Em: *Proceedings of Distributed Applications and Interoperable Systems*, 6th IFIP WG 6.1 International Conference, DAIS 2006. Volume 4025 of Lecture Notes in Computer Science. pg. 213–227. Bologna, Itália. Junho, 2006.

ALMEIDA, J.P.A.; IACOB, M.E.; JONKERS, H.; QUARTEL, D.A.C. *Platform-Independent Modelling of Service Infrastructure Components*, Freeband A-MUSE/D1.6, TI/RS/2005/078, Telematica Instituut, Enschede, Holanda, 2005.Disponível em: https://doc.telin.nl/dscgi/ds.py/Get/File-59319

BARBOSA, S. D. J.; SILVA, B. S. *Interação Humano-Computador.*[s.l.]Elsevier, 2010.

BARDRAM, J.E. The Java Context Awareness Framework (JCAF) – a service infrastructure and programming framework for context-aware applications.*Pervasive 2005*.Vol. 3468. 2005.

BELLAVISTA, P.; CORRADI, A.; FANELLI, M.; FOSCHINI, L.A survey of context data distribution for mobile ubiquitous systems.Em: *ACM Computing Surveys* (CSUR) 44, 4, Artigo 24, 45 páginas, Setembro 2012.

BERTRAN, B.; BRUNEAU, J.; CASSOU, D.; LORIANT, N.; BALLAND, E.; CONSEL, C. DiaSuite: A tool suite to develop Sense/Compute/Control applications.*Science of Computer Programming*, Volume 79, 1 Janeiro 2014, pg 39-51.

BOTÍA, J.A.; VILLA, A.; PALMA, J.T.; PEREZ, D.; IBORRA, E. Detecting domestic problems of elderly people: simple and unobstrusive sensors to generate the context of the attended. *First International Workshop on Ambient Assisted Living*.IWAAL, 2009.Vol. 5602 of LNCS. 2009.

BRAJDIC, A.; HARLE, R. 2013. Walk detection and step counting on unconstrained smartphones. *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing (UbiComp '13)*. ACM, New York, NY, USA, 225-234. 2013.

CARRIERO, N.; GELERNTER, D. Linda in context. *Communications of ACM*, ACM, New York, NY, USA, v. 32, n. 4, p. 444-458. 1989.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G.*Distributed Systems - Concepts and Design*. Ed. Addison-Wesley. 2012. ISBN: 978-0-13-214301-1.

DANIELE, L.M.; PIRES, L.F.; SINDEREN, M. VAN.An MDA-Based Approach for Behaviour Modelling of Context-Aware Mobile Applications, *Model Driven Architecture - Foundations and Applications, 5th European Conference*, ECMDA-FA 2009, pg 206-220, Enschede, Holanda, Junho 23-26, 2009.

DEURSEN, A. VAN; KLINT, P.; VISSER, J. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6):26-36, Junho 2000.

DEY, A.K. Understanding and Using Context.*Personal Ubiquitous Computing* Vol. 5, Ed. 1, pg. 4-7. Janeiro 2001.

DIRGAHAYU, T. Model-Driven Engineering ofWeb Service Compositions: A Transformation from ISDL to BPEL. Dissertação de Mestrado. University of Twente, Holanda. 2005

ECLIPSE PROJECT. *The Eclipse Modeling Framework (EMF) Overview*.Disponível em: <http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.emf.doc%2Freferences%2Foverview%2FEMF.html> Acesso em: 27 set 2014. 2005

FEILKAS, M. *How to represent Models, Languages and Transformations?,* pg. 169-176. 2006.

FONTELES, A.S.; NETO, B.J.A.; MAIA, M.; VIANA, W.; ANDRADE, R.M.C.An adaptive context acquisition framework to support mobile spatial and context-aware applications. Em: *Proceedings of the 12th international conference on Web and Wireless Geographical Information Systems* (W2GIS'13). Springer-Verlag, Berlin, Heidelberg, pg. 100-116.

FONTELES, A.S. *Um Framework para Aquisição Adaptativa e Fracamente Acoplada de Informação Contextual para Dispositivos Móveis*. Dissertação de Mestrado, Departamento de Computação, Universidade Federal do Ceará, Fortaleza, CE. 2013.

GRIGORAS, D. Challenges to the Design of Mobile Middleware Systems. Em: *International Symposium on Parallel Computing in Electrical Engineering*. PAR ELEC 2006., vol. 14, no. 19, pg. 13-17, Setembro, 2006.

HENRICKSEN, K.; INDULSKA, J.; MCFADDEN, T.; BALASUBRAMANIAM, S. Middleware for distributed context-aware systems. Em: *Proceedings of the 2005 Confederated international conference on On the Move to Meaningful Internet Systems* - Volume Part I (OTM'05). pg. 846-863. Ed. Springer-Verlag, Berlin, Alemanha. 2005.

HOYOS, J.R.; MOLINA, J.G.; BLAYA, J.A.B. MLContext: A Context-Modeling Language for Context-Aware Systems. Em: *Electronic Communications of the EASST* (ECEASST), Vol. 28., 2010.

HOYOS, J.R.; MOLINA, J.G.; BLAYA, J.A.B.A domain-specific language for context modeling in context-aware systems.*Journal of Systems and Software*, Volume 86, Issue 11, Novembro 2013, pgs 2890-2905.

KAKOUSIS, K.; PASPALLIS, N.; PAPADOPOULOS, G. A.A survey of software adaptation in mobile and ubiquitous computing.*Enterp. Inf. Syst*. 4, 4, pg. 355-389. Novembro 2010.

KLEPPE, A. G.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Ed. Addison-Wesley Longman Publishing Co., Inc., Boston, USA. 2003.

JOUAULT, F.; BÉZIVIN, J.; KURTEV, I. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. *Fifth International Conference on Generative Programming and Component Engineering* (GPCE'06), pgs 249–254. 2006.

LI F.; SEHIC, S.; DUSTDAR, S. COPAL: An adaptive approach to context provisioning, *2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications* (WiMob). pg.286-293, 11-13 Outubro 2010.

LIMA, F.F.P. *SysSu - Um Sistema de Suporte para Computação Ubíqua*. Dissertação de Mestrado, Departamento de Computação, Universidade Federal do Ceará, Fortaleza, CE. 2011

LIMA, F.F.P.; ROCHA, L.S.; MAIA, P.H.M.; ANDRADE, R.M.C. Uma Arquitetura Desacoplada e Interoperável para Coordenação em Sistemas Ubíquos. Em: *V Simpósio Brasileiro de Componentes, Arquitetura e Reuso de Software* (SBCARS). São Paulo, Setembro, 2011.

LUCRÉDIO, D.*Uma Abordagem Orientada a Modelos para Reutilização de Software*. Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, Julho 2009.

LUCRÉDIO, D.; BITTAR, T.J.; FORTES, R.P.M. Minicurso: Desenvolvimento web orientado a modelos: Conceitos, ferramentas e técnicas. Em: XV Simpósio Brasileiro de Sistemas Multimídia e Web (WEBMEDIA), 2009, Fortaleza - CE. *Anais Simpósio Brasileiro de Sistemas Multimídia e Web - Minicursos* (WEBMEDIA). Fortaleza/CE: Sociedade Brasileira de Computação, 2009. v. III. pg. 81-124.

MAIA, M.E.F.; FONTELES, A.; NETO, B.J.A; VIANA, W.; ANDRADE, R.M.C. LOCCAM - Loosely Coupled Context Acquisition Middleware, Em: *28th Symposium on Applied Computing* (SAC), Coimbra, Portugal. Março, 2013.

MAIA, M.E.F.;ROCHA, L.S.;ANDRADE, R.M.C. Requirements and challenges for building service-oriented pervasive middleware, Em: *Proceedings of the 2009 international conference on Pervasive services* (ICPS '09). pg. 93-102. ACM, New York, NY, EUA, 2009.

NIEMELÄ, E.; LATVAKOSKI, J. Survey of requirements and solutions for ubiquitous software. Em: *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia* (MUM '04). pg. 71-78. ACM, New York, NY, EUA. 2004.

OMG.(2003). *MDA guide version 1.0.1.Technical Report omg/2003-06-01, Object Management* Group, 2003.Disponível em: <http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf>.

OPEN SIGNAL. *Android Fragmentation Visualized (August 2014)*. Agosto de 2014. Disponível em <http://opensignal.com/reports/2014/android-fragmentation/>

SANTOS, A.C.; DINIZ, P.C.; CARDOSO, J.M.P.; FERREIRA, D.R. A Domain-Specific Language for the Specification of Adaptable Context Inference. *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing (EUC)*, pg.268-273, 24-26 Outubro, 2011.

SANTOS, A.C.;CARDOSO, J.M.P.;DINIZ, P.C.; FERREIRA, D.R. Specifying Adaptations through a DSL with an Application to Mobile Robot Navigation.*2nd Symposium on Languages, Applications and Technologies*.SLATE 2013, pg. 219-234. 2013

SCHILIT, B.N.; THEIMER, M.M. Disseminating active map information to mobile hosts.*IEEE Network*, vol.8, no.5, pg.22-32, Setembro-Outubro. 1994

SCHMIDT, D.C. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, vol.39, nº.2, pg. 25-31, Fevereiro, 2006.

SEHIC, S.; LI F.; DUSTDAR, S. COPAL-ML: a macro language for rapid development of context-aware applications in wireless sensor networks. *Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications* (SESENA '11). ACM, New York, NY, USA, 1-6. 2011.

STAHL, T. e VÖLTER, M.; *Model-driven software development: technology, engineering*, management, John Wiley & Sons Ltd, 2006.

TAKAHASHI, H.; SUGANUMA, T.; SHIRATORI, N. AMUSE: An Agent-based Middleware for Context-aware Ubiquitous Services.*11th International Conference on Parallel and Distributed Systems* (ICPADS'05), pG.743-749, 2005

VIANA, W.*Mobilité et sensibilité au contexte pour la gestion de documents multimédias personnels : CoMMedia*. Tese de Doutorado. University of Grenoble. 2010.

VIANA, W.; BRAGA, R.; LEMOS, F.; SOUZA, J. O.; CARMO, R.; ANDRADE, R.M.C.; MARTIN, H. Mobile Photo Recommendation and Logbook Generation From Context-Tagged Images.*MultiMedia*, IEEE , vol.PP, no.99, pp.1,1, 0. 2014.

WEISER, M.The computer for the 21st century.*Scientific American*, pg. 94–104.1991.

# Apêndice A – Lista de CACs Existentes no Repositório

| *Context Key* | **Descrição** |
|---|---|
| context.ambient.calendar | Retorna a data do dia atual |
| context.ambient.dayofweek | Retorna o dia da semana |
| context.ambient.gravity | Retorna o valor da aceleração da gravidade no local nos eixos X, Y e Z em metros por segundo ao quadrado (m/s²) |
| context.ambient.gravity.axisx | Retorna o valor da aceleração da gravidade no local nos eixo X em metros por segundo ao quadrado (m/s²) |
| context.ambient.gravity.axisy | Retorna o valor da aceleração da gravidade no local nos eixo Y em metros por segundo ao quadrado (m/s²) |
| context.ambient.gravity.axisz | Retorna o valor da aceleração da gravidade no local nos eixo Z em metros por segundo ao quadrado (m/s²) |
| context.ambient.light | Retorna o valor da luminosidade ambiente em lux (l) |
| context.ambient.magnetic | Retorna o valor do campo magnético no local nos eixos X, Y e Z em micro-Tesla (µT) |
| context.ambient.magnetic.axis x | Retorna o valor do campo magnético no local no eixo X em micro-Tesla (µT) |
| context.ambient.magnetic.axis y | Retorna o valor do campo magnético no local no eixo Y em micro-Tesla (µT) |
| context.ambient.magnetic.axis z | Retorna o valor do campo magnético no local no eixo Z em micro-Tesla (µT) |
| context.ambient.pressure | Retorna o valor da pressão atmosférisca em hectopascal (hPa) |
| context.ambient.relativehumidity | Retorna o valor da umidade relativa do ar no ambiente em percentual |
| context.ambient.temperature | Retorna o valor da temperatura ambiente em Celsius (ºC) |

| | |
|---|---|
| context.ambient.temperature.fahrenheit | Retorna o valor da temperatura ambiente na escala Fahrenheit (ºF) |
| context.ambient.temperature.kelvin | Retorna o valor da temperatura ambiente na escala Kelvin (K) |
| context.ambient.weather | Retorna informações sobre o clima na forma de um XML vindo de um *Web Service* |
| context.ambient.weather.fortaleza | Retorna informações sobre o clima de Fortaleza na forma de um XML vindo de um *Web Service* |
| context.device.acceleration | Retorna o valor da aceleração do dispositivo nos eixos X, Y e Z em metros por segundo ao quadrado (m/s²) |
| context.device.acceleration.axisx | Retorna o valor da aceleração do dispositivo no eixo X em metros por segundo ao quadrado (m/s²) |
| context.device.acceleration.axisy | Retorna o valor da aceleração do dispositivo no eixo Y em metros por segundo ao quadrado (m/s²) |
| context.device.acceleration.axisz | Retorna o valor da aceleração do dispositivo no eixo Z em metros por segundo ao quadrado (m/s²) |
| context.device.acceleration.resulting | Retorna o valor do módulo do vetor resultante da aceleração do dispositivo em metros por segundo ao quadrado (m/s²) |
| context.device.gpslocation | Retorna o valor da localização aproxima do dispositivo através do GPS |
| context.device.gyroscope | Retorna o valor da velocidade angular do dispositivo em torno dos eixos X, Y e Z em radianos por segundo (rad/s) |
| context.device.gyroscope.axisx | Retorna o valor da velocidade angular do dispositivo em torno do eixo X em radianos por segundo (rad/s) |
| context.device.gyroscope.axisy | Retorna o valor da velocidade angular do dispositivo em torno do eixo Y em radianos por segundo (rad/s) |
| context.device.gyroscope.axisz | Retorna o valor da velocidade angular do dispositivo em torno do eixo Z em radianos por segundo (rad/s) |
| context.device.linearacceleration | Retorna o valor da aceleração linear do dispositivo (aceleração excluindo a gravidade) nos eixos X, Y e Z em metros por segundo ao quadrado (m/s²) |

| | |
|---|---|
| context.device.linearacceleration.axisx | Retorna o valor da aceleração linear do dispositivo (aceleração excluindo a gravidade) no eixo X em metros por segundo ao quadrado (m/s²) |
| context.device.linearacceleration.axisy | Retorna o valor da aceleração linear do dispositivo (aceleração excluindo a gravidade) no eixo Y em metros por segundo ao quadrado (m/s²) |
| context.device.linearacceleration.axisz | Retorna o valor da aceleração linear do dispositivo (aceleração excluindo a gravidade) no eixo Z em metros por segundo ao quadrado (m/s²) |
| context.device.networklocation | Retorna o valor da localização aproxima do dispositivo através do Wi-Fi |
| context.device.proximity | Retorna o valor fornecido pelo sensor de proximidade do dispositivo medido em centímetros (cm) |
| context.device.random | Retorna o valor de um número randômico gerado automaticamente a cada dez segundos |
| context.device.wifi | Retorna o valor das informações contextuais sobre as redes Wi-fi próximas ao dispositivo |

# Apêndice B – Questionário Pré-Experimento da Avaliação de Usabilidade do CRITiCAL

**Nome:**

**Q1. Eu tenho um alto grau de conhecimento em programação em Java**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q2. Eu tenho um alto grau de experiência em desenvolvimento de programas utilizando a plataforma Eclipse**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q3. Eu tenho um alto grau de conhecimento no desenvolvimento de aplicações móveis e sensíveis ao contexto**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q4. Eu tenho um alto grau de conhecimento no desenvolvimento de aplicações para a plataforma Android**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q5. Eu tenho um alto grau de conhecimento do paradigma de desenvolvimento de software orientado a modelos**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q6. Eu tenho um alto grau de experiência no desenvolvimento de aplicações Android com acesso a sensores**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q7. Eu tenho um alto grau de conhecimento no *middleware*LoCCAM**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q8. Grau de escolaridade:**

1 - Graduando

2 - Graduado

3 - Cursando especialização

4 - Especialista

5 - Mestrando

6 - Mestre

7 - Doutorando

8 - Doutor

# Apêndice C – Respostas dos Voluntários ao Exercício da Tarefa 4 da Avaliação de Usabilidade do CRITiCAL

**Tabela Apêndice B.1 – Respostas dos Voluntários**

| Voluntário | Resposta Regra 01 | Resposta Regra 02 |
|---|---|---|
| 01 | (A regra ainda não está ativa) A regra ativa, quando: Enquanto a luz do ambiente > 10 E Enquanto a temperatura do ambiente > 280 (kelvin) E Enquanto a umidade do ambiente <50 | (A regra está ativa) A regra ativa, quando: Quando aceleração X do dispositivo < 10 OU Quando o dia da semana for sexta-feira |
| 02 | Enquanto a intensidade da luz for maior que 10 e a temperatura ambiente em Kelvin for maior que 280 e a umidade relativa do ar for menor que 50, a ação notifyapp é ativada. | Enquanto a acerelação do eixo X for menor que 10 ou o dia da semana for igual a sexta-feira será executada a ação ExibirNaTela. |
| 03 | A regra 1 é ativada quando as seguintes condições são verdadeiras: enquanto o valor de luz ambiente for maior que 10,0 e enquanto o valor de temperatura for maior que 280K e enquanto o valor de umidade relativa for menor que 50,0. Os valores são obtidos a partir do device. Quando a regra é ativada uma notificação é enviada com a string "Regra Ativada". | A regra 2 é ativada quando as seguintes condições são verdadeiras: enquanto o valor de aceleração do eixo X for menor que 10,0 ou quando o dia da semana for igual a Friday. Os valores são obtidos a partir do device e do ambient. Quando a regra é ativada é mostrado na tela a string "Regra Ativada". |
| 04 | Enquanto a Condição Luz for maior que 10 Enquanto a Condição Temperatura do tipo kelvin for maior que 280 Enquanto a Condição Umidade for menor que 50 As condições são um "E" Não é um evento A ação é notifyApp | Enquanto a aceleração do dispositivo for menor que 10 ou o dia da semana for igual a sexta-feira, exibe na tela da aplicação. |
| 05 | executar uma ação enquanto a intensidade da luz for maior que 10 e enquanto a temperatura for maior que 280 graus kelvin e enquanto a umidade for menor que 50. | Enquanto aceleração no eixo x menor que 10 ou o dia da semana for igual a Friday. Exibição na tela. |

| | | |
|---|---|---|
| 06 | Enquanto a medida de luminosidade for maior 10, temperatura em kelvin for maior que 280 e umidade for menor que 50 ative a ação notifyapp | Quando a aceleração do eixo x for menor que 10 e o dia da semana for sexta-feira ative a ação exibirnatela |
| 07 | while (luz > 10 and temperatura.Tokelvin () and umidade < 50) regra atividade | Regra de Exemplo - AceleraçaoEixoX: quando a aceleração do dispositivo no eixo X for menor que 10 OU - DiaDaSemana: se o dia da semana for Sexta Vai aparecer na tela a mensagem "Regra Ativada" (ExibirNaTela) => É um evento |
| 08 | Regra de Exemplo - Condição Luz: Enquanto a luz do ambiente for maior que 10 E - Condição Temperatura: Enquanto a temperatura ambiente em Kelvin for maior que 280 E - Condição Umidade: Enquanto a umidade relativa do ambiente for maior que 50 Vai aparecer no log a mensagem "Regra Ativada" (notifyapp) => Não é um evento | Quando a aceleração do eixo x estiver meno que 10 OU o dia da semana for sexta então regra ativada |
| 09 | Primeiro condição: maior que 10 - Informação contextual: luz - Derivação: ambiente - Retorno: double Segunda condição: maior que 280 - Informação contextual: temperatura - Derivação: ambiente - Retorno: double - Qualifer: Kelvin Terceira condição: menor que 50 - Informação contextual: umidade relativa - Derivação: ambiente - Retorno: double | Primeiro condição: menor que 10 - Informação contextual: aceleração - Derivação: device - Retorno: double - Qualifer: eixo X Primeiro condição: igual ao friday - Informação contextual: dia da semana - Retorno: String |
| 10 | Existem três condições na RegraDeExemplo. A primeira condição verifica enquanto a Luz do AMBIENT está maior que 10. A segunda condição verifica enquanto a Temperatura do AMBIENT está maior que 280 kevin. A terceira e última condição verifica enquanto a Umidade é menor que 50. Esses dados irão ser aplicados na atividade notifyapp. | Existem três condições na RegraDeExemplo. A primeira condição verifica enquanto a AceleraçaoEixoX do DEVICE está menor que 10. A segunda condição verifica se o DiaDaSemana do AMBIENT é igual a "Friday". Esses dados irão ser aplicados na atividade ExibirNaTela. |
| 11 | Enquanto a luz do ambiente for maior que 10 e a temperatura do ambiente for maior que 280 Kelvin e enquanto a umidade do ambiente for menor que 50, notifique a aplicação. | O diagrama ilustra uma regra exemplo composta por duas condições: 1. Se o acelerometro do dispositivo retornar um valor menor que 10; 2. Se é sexta-feira Uma vez que as duas condições são devidamente atendidas, a regra é ativada e algo será |

| | | |
|---|---|---|
| | | renderizado na tela. |
| 12 | O diagrama ilustra uma regra exemplo composta por três condições: 1. Se luz maior que 10 2. Se temperatura maior que 280 (em Kelvins) 3. Se a umidade relativa for menor que 50 Uma vez que as três condições são devidamente atendidas, a regra é ativada e a aplicação notificada. | Executar uma ação quando a aceleração no eixo X for menor que 10 ou quando o dia da semana for sexta-feira. |
| 13 | Aplicação de controle de luz, temperatura e umidade de um ambiente. Enquanto a luz maior do que 10 lux, e temperatura maior que 280 Kelvin e a umidade tiver 50 unidades. | Quando a Condição Aceleração do Eixo X menor que 10 Ou quando o dia da semana é igual a Friday Exibir na Tela É um evento |
| 14 | enquanto a luminosidade estiver maior que 10 E a temperatura estiver maior que 280 E a umidade relativa estiver maior que 50 então regra ativada. | while (aceleração.axisX() < 10) and dayofweek = friday regra ativada |

# Apêndice D – Questionário Pós-Experimento da Avaliação de Usabilidade do CRITiCAL

**Nome:**

**Q1. Eu usaria esta ferramenta com frequência quando desenvolvesse aplicações que utilizem o LoCCAM**

1 - Discordo completamente
2 - Discordo
3 - Neutro
4 - Concordo
5 - Concordo completamente

**Q2. Eu achei que esta ferramenta é fácil de utilizar**

1 - Discordo completamente
2 - Discordo
3 - Neutro
4 - Concordo
5 - Concordo completamente

**Q3. Eu não achei que necessitaria do suporte de um técnico para me tornar hábil a utilizar esta ferramenta**

1 - Discordo completamente
2 - Discordo
3 - Neutro
4 - Concordo
5 - Concordo completamente

**Q4. Eu achei que as várias funções desta ferramenta eram bem integradas**

1 - Discordo completamente
2 - Discordo
3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q5. Eu achei esta ferramenta consistente**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q6. Eu considerei que a maioria das pessoas aprenderiam a utilizar esta ferramenta de maneira rápida**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q7. Eu me senti muito confiante ao utilizar a ferramenta**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q8. Eu não precisei aprender muitas coisas novas antes que pudesse utilizar direito esta ferramenta**

1 - Discordo fortemente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo fortemente

**Q9. Você tem alguma sugestão, elogio ou crítica em relação à ferramenta como um todo?**

# Apêndice E – Questionário Comparativo entre o desenvolvimento com o CRITiCAL e utilizando o LoCCAM manualmente

**Nome:**

**Q1. Eu considerei o desenvolvimento com a CRITiCAL de fácil aprendizado**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q2. Eu considerei os métodos de comunicação entre aplicação e middleware (read, put, take, subscribes) mais simples e fáceis de entender e aplicar do que utilizando apenas o LoCCAM**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q3. Eu considerei a modelagem das informações e das regras contextuais perfeitamente claras**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q4. Eu considerei a forma de configuração e instalação do LoCCAM e dos CACS utilizando a CRITiCAL melhor do que a versão manual**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q5. As abstrações utilizadas na linguagem para representar os conceitos de regras e informações contextuais são satisfatórias**

1 - Discordo completamente

2 - Discordo

3 - Neutro

4 - Concordo

5 - Concordo completamente

**Q6. Você tem alguma sugestão, elogio ou crítica em relação à ferramenta como um todo?**