UNIVERSIDADE FEDERAL DO MARANHÃO

Programa de Pós-Graduação em Ciência da Computação

Luís Eduardo Costa Laurindo

Especificação e Monitoramento de Requisitos de Qualidade de Contexto para Aplicações de Fenotipagem Digital

São Luís
2023

Luís Eduardo Costa Laurindo

Especificação e Monitoramento de Requisitos de Qualidade de Contexto para Aplicações de Fenotipagem Digital

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Ciência da Computação, ao Programa de Pós-Graduação em Ciência da Computação, da Universidade Federal do Maranhão.

Programa de Pós-Graduação em Ciência da Computação

Universidade Federal do Maranhão

Orientador: Prof. Dr. Francisco José da Silva e Silva

Coorientador: Prof. Dr. Luciano Reis Coutinho

São Luís - MA

2023

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Laurindo, Luís Eduardo Costa.
Especificação e Monitoramento de Requisitos de Qualidade de Contexto para Aplicações de Fenotipagem Digital / Luís Eduardo Costa Laurindo. - 2023.
87 f.

Coorientador(a): Luciano Reis Coutinho.
Orientador(a): Francisco José da Silva e Silva.
Dissertação (Mestrado) - Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, São Luís, 2023.

1. Aquisição e Distribuição. 2. Especificação. 3. Fenotipagem Digital. 4. Monitoramento. 5. Qualidade de Contexto (QoC). I. Coutinho, Luciano Reis. II. da Silva e Silva, Francisco José. III. Título.

Luís Eduardo Costa Laurindo

Especificação e Monitoramento de Requisitos de Qualidade de Contexto para Aplicações de Fenotipagem Digital

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Ciência da Computação, ao Programa de Pós-Graduação em Ciência da Computação, da Universidade Federal do Maranhão.

Trabalho aprovado. São Luís - MA, 17 de Agosto de 2023:

Prof. Dr. Francisco José da Silva e Silva
Orientador
Universidade Federal do Maranhão

Prof. Dr. Luciano Reis Coutinho
Coorientador
Universidade Federal do Maranhão

Prof. Dr. Ariel Soares Teles
Examinador Interno
Universidade Federal do Maranhão

Prof. Dr. André Castelo Branco Soares
Examinador Externo
Universidade Federal do Piauí

São Luís - MA
2023

Aos meus pais que sempre me
apoiam em todas as minhas decisões

Agradecimentos

Primeiramente agradeço a Deus por me conceder o dom da vida e me proporcionar momentos como esse.

Ao meu orientador, o Prof. Francisco José da Silva e Silva, e ao meu coorientador, o Prof. Luciano Reis Coutinho pelos direcionamentos e ensinamentos ao longo deste mestrado, por estarem sempre presente em todas as etapas. Pela paciência nos momentos em que tive dificuldade, e pelo apoio nos momentos em que duvidei da minha capacidade. Sou muito grato a vocês dois.

Agradeço a minha família em especial a meu pai Cláudio Laurindo e, minha mãe Francisca Laurindo que sempre fizeram de tudo por mim. Aos meus padrinhos, Socorro Gomes e Pedro Gomes, por me receber em sua casa por um período de 4 anos na época da graduação, e me oferecer todo apoio enquanto estive longe de casa. A minha namorada Victória Sawanna por compreender os momentos de ausência, e por me apoiar em todas as minhas decisões, sei que não foi um período fácil.

A todos os integrantes do LSDi, em especial aos meus amigos, Ivan Rodrigues, André Luiz, Bruno Roberto, Rodrigo Sirqueira, Joeckson dos Santos que sempre me apoiaram e me ajudaram de alguma forma com a pesquisa. Aos meus amigos de turma, Jovennilton Soares, Lucélia Lima e Maurício Morais que também contribuíram para realização dessa pesquisa.

Aos meus amigos, Matias Romário e novamente ao Ivan Rodrigues, que me ajudaram a ingressar no mestrado. Ao meu amigo, Rodrigo Alves por todo incentivo, por me ouvir e aconselhar nos momentos difíceis.

Aos meus orientadores da graduação, o Prof. Francisco José de Araújo e, o Prof. Ricardo Moura Sekeff Budaruiche por me mostrar as oportunidades e me motivar para a realização do mestrado. Foi por meio deles que tive o meu primeiro contato com a pesquisa.

Agradeço à Universidade Federal do Maranhão (UFMA) pela oportunidade de fazer a pós-graduação. O apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES - Código Financeiro 001); INCT da Internet do Futuro para Cidades Inteligentes (CNPq 465446/2014-0, CAPES 88887.136422/2017-00, e FAPESP 14/50937-1 e 15/24485-9); Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (311608/2017-5, 420907/2016-5, 312324/2015-4); e a Agência de Amparo à Pesquisa do Estado do Maranhão – FAPEMA (via outorga BM QUOTA UFMA - CONVÊNIO Nº 01/2021 FAPEMA/UFMA) pelo apoio financeiro durante o mestrado.

Agradeço a todos que de alguma forma contribuíram para realização deste mestrado!

"Paciência e perseverança tem o efeito mágico
de fazer as dificuldades desaparecerem
e os obstáculos sumirem."

(John Quincy Adams)

Resumo

As aplicações de fenotipagem digital utilizam dados de sensores de dispositivos digitais pessoais (*smartphones*, *smartbands*) para quantificar o fenótipo humano momento a momento no nível individual *in-situ*. Alguns aspectos podem degradar a Qualidade do Contexto (QoC) da aplicação, como informações dos sensores imprecisas, tecnologias de comunicação sem fio utilizadas na aquisição e distribuição de informações, problemas de escalabilidade e conexão intermitente devido à mobilidade do usuário. Garantir a qualidade e distribuição das informações de contexto utilizadas é requisito essencial no domínio destas aplicações. Portanto, este estudo concebeu um processo para incorporar requisitos de QoC em aplicações para fenotipagem digital. O processo consiste em cinco etapas, a saber: (i) realizar a especificação dos requisitos de QoC por meio de um metamodelo; (ii) a geração de código alvo por meio de um mecanismo de transformação que recebe como entrada a especificação dos requisitos; (iii) a incorporação na aplicação dos códigos fonte gerados para garantir os requisitos de QoC; (iv) monitoramentos dos parâmetros de QoC; (iv) visualização do resultado do monitoramento por meio de *Dashboards* que visa fornecer ao desenvolvedor um ambiente no qual seja possível acompanhar o nível de qualidade das instâncias da aplicação. Para avaliar a solução proposta realizou-se o desenvolvimento de um estudo de caso visando avaliarmos a aplicabilidade do processo proposto em uma aplicação típica da fenotipagem digital. Adicionalmente, foram realizados experimentos para analisar o desempenho da aplicação com e sem os mecanismos de QoC, para avaliarmos o impacto da QoC durante a execução da mesma. Por meio da aplicação do processo proposto no estudo de caso desenvolvido, observamos a facilidade para incorporar requisitos de QoC em aplicações para fenotipagem digital. Ao realizar os experimentos de QoC, foi possível verificar que a aplicação instrumentada com os mecanismos de QoC garante a qualidade e a entrega das informações que serão utilizadas para inferência de situações de interesse.

Palavras-chave: Fenotipagem Digital, Aquisição e Distribuição, Qualidade de Contexto (QoC), Especificação, Transformação, Monitoramento.

Abstract

Digital phenotyping applications use sensor data from personal digital devices (smartphones, smart bands) to quantify the moment-to-moment human phenotype at the individual in-situ level. Some aspects can degrade the Quality of Context (QoC) of applications, such as inaccurate sensor information, wireless communication technologies used in acquiring and distributing information, scalability problems, and intermittent connection due to user mobility. Ensuring the quality and distribution of the data used is an essential requirement in the domain of these applications. Therefore, this study conceived a process to incorporate QoC requirements in applications for digital phenotyping. The process consists of five steps, namely: (i) specifying the QoC requirements using a metamodel; (ii) target code generation through a transformation mechanism that receives the requirements specification as input; (iii) the incorporation in the application of the generated source codes to guarantee the QoC requirements; (iv) evaluation and monitoring of QoC parameters; (iv) visualization of the monitoring result through Dashboards, which aims to provide the developer with an environment in which it is possible to monitor the quality level of the application instances. A case study was developed to evaluate the applicability of the proposed process in a typical application of digital phenotyping. Additionally, experiments were carried out to analyze the application's performance with and without the QoC mechanisms to evaluate the impact of the QoC during its execution. By applying the process proposed in the developed case study, we observed the ease of incorporating QoC requirements in applications for digital phenotyping. When carrying out the QoC experiments, it was possible to verify that the application instrumented with the QoC mechanisms guarantees the quality and delivery of the information that will be used for the inference of situations of interest.

Keywords: Digital Phenotyping, Acquisition and Distribution, Quality of Context (QoC), Specification, Transformation, Monitoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AAL	*Ambient Assisted Living*
API	*Application Programing Interface*
AQL	*Acceleo Query Language*
BT	*Bluetooth Classic*
BLE	*Bluetooth Low Energy*
CEP	*Processamento de Eventos Complexos*
CDDL	*Context Data Distribution Layer*
DSL	*Domain Specific Language*
EMF	*Eclipse Modeling Framework*
EPA	*Event Processing Agent*
EPL	*Event Processing Language*
EPN	*Event Processing Network*
FSML	*Framework-Specific Modeling Languages*
QoC	*Qualidade de Contexto*
QoI	*Qualidade da Informação*
QoS	*Qualidade de Serviço*
IoT	*Internet das Coisas*
JSON	*JavaScript Object Notation*

LNICST	*Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*
LSDi	*Laboratório de Sistemas Distribuídos Inteligentes*
MDE	*Engenharia Dirigida a Modelos*
M2C	*Model-to-Code*
M2M	*Model-to-Model*
M2T	*Model-to-Text*
M-Hub	*Mobile Hub*
MQTT	*Message Queuing Telemetry Transport*
MTL	*Model to Text Language*
RSL	*Revisão Sistemática da Literatura*
SDK	*Software Development Kit*
SMMS	*Sociability Monitoring Mobile System*
SQL	*Structured Query Language*

Sumário

1 Introdução

Atualmente há cerca de 6 bilhões de *smartphones* e 4 bilhões de dispositivos vestíveis no mundo (Statista, 2022; OMETOV et al., 2021). Estima-se que esse número deva aumentar, chegando a 7,51 bilhões e 7,10 bilhões em 2026, respectivamente. Esses dispositivos fazem parte do dia a dia das pessoas, e possuem um conjunto de sensores que capturam o contexto do usuário e do ambiente onde estão inseridos. Métodos computacionais podem utilizar essas informações do contexto para realizar inferências sobre aspectos sociais, comportamentais e cognitivos dos indivíduos (MOURA et al., 2020; SACCARO et al., 2021). Por exemplo, é possível identificar se o usuário está realizando alguma atividade física por meio de dados de localização e aceleração ou se está conversando com base nos dados captados pelo microfone do *smartphone*.

A Fenotipagem Digital é uma área de pesquisa que visa utilizar dados de contexto de dispositivos móveis e vestíveis para inferir o estado de saúde do indivíduo (TOROUS et al., 2016). A pesquisa em fenotipagem digital se concentra no desenvolvimento de novas soluções para complementar e ampliar as fontes tradicionais de dados clínicos. Torous et al. (2016) definem Fenotipagem Digital como a "quantificação momento a momento do fenótipo humano no nível individual *in-situ* usando dados de *smartphones* e outros dispositivos digitais pessoais".

Segundo Mendes et al. (2022a) o processo de fenotipagem digital começa com a coleta de dados brutos de sensores, sejam físicos (e.g., GPS, acelerômetro, frequência cardíaca) ou virtuais (e.g., chamadas telefônicas, tempo de tela ativado, aplicativos utilizados). Após a coleta dos dados brutos, eventos comportamentais e de saúde são inferidos. Eventos comportamentais representam ações realizadas pelo indivíduo (e.g., o intervalo de tempo em que ele socializa com a família). Os eventos de saúde representam o estado fisiológico do indivíduo com base nos sinais vitais (e.g., frequência cardíaca, pressão arterial, oxigenação sanguínea). Após a inferência desses eventos, os padrões comportamentais são obtidos (e.g., o padrão de mobilidade, sociabilidade e atividade física). Esses padrões referem-se a situações rotineiras dos indivíduos (e.g., o indivíduo dorme sempre, de segunda a sexta-feira, às 23 horas). Finalmente, esses padrões comportamentais são usados em aplicações de previsão e diagnóstico.

O desenvolvimento de aplicações de fenotipagem digital requer componentes responsáveis pela coleta de dados de contexto de sensores embarcados/conectados a dispositivos digitais pessoais. Além disso, os desenvolvedores devem implementar componentes de *software* para inferir eventos comportamentais e distribuir dados para servidores externos via infraestrutura de comunicação sem fio. Por fim, no servidor, é necessário implementar

componentes para gerenciamento de grandes volumes de dados e modelos de aprendizado de máquina para realização de análises com base nas informações coletadas.

As soluções de fenotipagem digital precisam garantir um nível aceitável de qualidade dos dados para obter maior precisão na tomada de decisão, especialmente quando aplicadas na área da saúde. Qualidade de Contexto (QoC) refere-se à composição da Qualidade da Informação (QoI) e Qualidade de Serviço (QoS) (BELLAVISTA et al., 2012). Com os recursos oferecidos pela QoC, é possível estabelecer contratos de qualidade entre provedores de serviço e consumidores, selecionar fontes de melhor contexto, aumentar a eficiência das aplicações, otimizando o consumo de energia e largura de banda, adequando a frequência de distribuição das informações e melhorando a qualidade da experiência do usuário (BUCHHOLZ; KÜPPER; SCHIFFERS, 2003).

Na literatura, pode-se encontrar diversos parâmetros relacionados à QoC para especificar os requisitos de qualidade da aplicação (GOMES et al., 2017; JAGARLAMUDI et al., 2022). Por exemplo, o parâmetro `Age` pode ser utilizado para definir que informações de contexto desatualizadas não sejam utilizadas pela aplicação para uma tomada de decisão em tempo real. Já o parâmetro `Refresh Rate` pode ser definido para especificar a frequência desejada para a distribuição das informações de modo a diminuir o fluxo de dados enviado.

As aplicações de fenotipagem digital utilizam-se de tecnologias (e.g., sensores, rede sem fio) e são executadas em ambientes (e.g., o usuário está em um local com uma cobertura de internet limitada) que podem degradar a QoC da aplicação. Primeiro, elas utilizam dados de sensores que podem gerar leituras imprecisas e errôneas (CHO et al., 2021). Em segundo lugar, lidam com uso de rede de comunicação sem fio e mobilidade do usuário que resultam em perda de dados. Por fim, uma infraestrutura de fenotipagem digital permite o monitoramento de um conjunto de indivíduos, podendo ocorrer problemas relacionados à escalabilidade. Portanto, é necessário especificar parâmetros de QoC para atender aos requisitos de cada aplicação, a fim de garantir um nível adequado de qualidade. Também é fundamental monitorar a conformidade com esses requisitos para sanar possíveis problemas que degradam a QoC da aplicação.

A fenotipagem digital é uma área de pesquisa bastante ativa atualmente (MOURA et al., 2020; MENDES et al., 2022a). Diversas plataformas de *middleware* foram concebidas com o intuito de facilitar o desenvolvimento de aplicações de fenotipagem digital, tais como (AHARONY et al., 2011; SCHUELLER et al., 2014; FERREIRA; KOSTAKOS; DEY, 2015; TOROUS et al., 2016; XIONG et al., 2016; MENDES et al., 2022b). Ao projetar essas aplicações, é necessário desenvolver ou utilizar plataformas de *software* para realizar a aquisição, inferência e distribuição de informações de contexto. No entanto, o tratamento de QoC na área de fenotipagem digital ainda é incipiente na literatura, principalmente em plataformas de *middleware* voltadas para esta área. Portanto, ao considerar a importância e

os requisitos dessas aplicações, este estudo concebe um processo para incorporar requisitos de QoC, e dispõe de mecanismos para especificar e monitorar parâmetros de QoC relevantes no domínio das aplicações de fenotipagem digital.

1.1 Objetivos

O objetivo geral desta pesquisa de mestrado é a concepção de um processo composto por etapas que permitam a especificação dos requisitos, geração de código que implementam os mecanismos de QoC e a sua incorporação a aplicação e mecanismos para o monitoramento da QoC durante a execução da aplicação.

Para tanto, consideram-se os seguintes objetivos específicos:

- Mapear o estado da arte com relação a plataformas de *middleware* e aplicações concebidas que realizam a coleta e distribuição de informações de contexto utilizadas no âmbito da fenotipagem digital em saúde.
- Conceber um metamodelo para especificação de parâmetros de QoC de acordo com requisitos da aplicação.
- Desenvolver um mecanismo para realizar a transformação da especificação de requisitos de QoC em códigos fonte para serem incorporados na aplicação pelo desenvolvedor no processo de desenvolvimento.
- Desenvolver mecanismos para realizar o monitoramento do atendimento dos parâmetros de QoC especificados pelo desenvolvedor.
- Conduzir uma avaliação experimental que permita comprovar a importância da incorporação de requisitos de QoC em aplicações para fenotipagem digital utilizando fluxo de dados reais.

1.2 Organização do Trabalho

Este trabalho está estruturado da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica onde aborda os conceitos de Qualidade de Contexto, Processamento de Eventos Complexos, a plataforma de *middleware* M-Hub/CDDL utilizada como base para o desenvolvimento da aplicação do estudo de caso para avaliar a solução proposta., Linguagens de Modelagem de Domínio Específico, e por fim, os conceitos sobre Transformações entre Modelos.

- O Capítulo 3 são apresentados os trabalhos relacionados referente ao desenvolvimento e utilização de plataformas de *softwares* para conceber aplicações para fenotipagem digital .
- O Capítulo 4 apresenta a solução proposta, na qual consiste em um processo definido para incorporar requisitos de QoC em aplicações para fenotipagem digital.
- O Capítulo 5 apresenta o desenvolvimento de uma aplicação intitulada LSDi mHealth como estudo de caso, e detalha a utilização do processo para incorporação de requisitos de QoC na aplicação.
- O Capítulo 6 apresentada avaliação experimental do uso de QoC no estudo de caso desenvolvido.
- O Capítulo 7 apresenta as conclusões, trabalhos futuros e os artigos científicos desenvolvidos.

2 Fundamentação Teórica

Nesta seção são apresentados os principais conceitos das temáticas e tecnologias que foram necessárias para a concepção deste estudo. São discutidos tópicos relacionados a Qualidade de Contexto, Processamento de Eventos Complexos, a plataforma de *middleware* M-Hub/CDDL utilizada para concepção de aplicações móveis no contexto de Internet das Coisas, Linguagens de Modelagem de Domínio Específico, e por fim, os conceitos sobre Transformações entre Modelos.

2.1 Qualidade de Contexto

Contexto é a informação usada para caracterizar a situação de uma entidade (por exemplo, pessoas, objetos ou lugares) que influencia as decisões de um agente (BOLCHINI et al., 2009). A primeira definição de Qualidade de Contexto (QoC) foi concebida por (BUCHHOLZ; KÜPPER; SCHIFFERS, 2003). Eles definem como qualquer informação que descreve a qualidade de uma informação de contexto. Em outra definição, (MANZOOR; TRUONG; DUSTDAR, 2014) conceituam qualidade de contexto como o nível de conformidade do contexto com as exigências de um consumidor de contexto. Por exemplo, uma aplicação que requisita dados de localização de um usuário pode definir como regra apenas receber os dados com um determinado valor de acurácia, ou uma aplicação no âmbito da saúde pode requerer apenas dados de um sensor de frequência cardíaca que possui um valor de acurácia dentro do nível de qualidade exigido pela aplicação. A seguir são descritos apenas alguns dos diversos parâmetros presentes na literatura utilizados para quantificar o grau de qualidade da informação, além de permitir a definição da qualidade do serviço de distribuição (GOMES et al., 2017; JAGARLAMUDI et al., 2022).

A Acurácia (`Accuracy`) representa uma estimativa de quão próxima a informação de contexto está do valor real. O fabricante do dispositivo normalmente fornece o valor de precisão quando a informação vem de um sensor físico. A Confiança (`Confidence`) estima o grau de certeza da informação fornecida pela fonte de informação. O Intervalo de Medição (`Measurement Interval`) indica o intervalo de tempo entre leituras sucessivas. O Atraso (`Delay`) representa o tempo decorrido entre o envio da mensagem pelo produtor e sua chegada ao consumidor. Mesmo sendo um parâmetro de QoS, é possível inserir o tempo computado nos dados de contexto como uma metainformação relacionada à QoI. Completude (`Completeness`) indica quão completa é a informação de contexto recebida por um consumidor. Pode ser calculado a partir da razão entre a soma dos pesos dos atributos disponíveis e a soma dos pesos dos atributos requeridos pelo consumidor. O Tempo de Validade (`Validity Time`) indica a validade da informação. O produtor da

informação pode especificar a validade da informação conforme ele a produziu. No entanto, o consumidor pode decidir se aceita ou não o prazo de validade especificado pelo produtor. A Idade (`Age`) também pode ser utilizada para verificar o tempo de validade da informação. Indica a diferença entre o instante atual e o tempo de medição da informação. Por fim, o Tempo Total de Entrega (`Total Delivery Time`) indica o tempo desde a mensuração da informação até sua entrega ao consumidor.

Para definir o nível de qualidade do serviço de distribuição de informações, temos a Confiabilidade (`Reliability`) que determina se o serviço de distribuição deve adotar a política de melhor esforço (`best-effort`), quando não há garantia de entrega de informação, ou utilizar `delivery` e retransmissão se a informação de contexto não for entregue ao destinatário. A Taxa de atualização (`Refresh Rate`) permite que o consumidor defina com que frequência as informações de contexto devem ser recebidas, independentemente da frequência com que os dados são produzidos. O Tempo de Entrega (`Deadline` indica o tempo máximo que um consumidor está disposto a esperar por uma informação. O Controle de Latência (`Latency Budget`) define um atraso adicional ao produtor e consumidor da informação. Ao definir um atraso, as mensagens são agrupadas em uma fila e enviadas ou recebidas em uma única rajada. O Histórico (`History`) permite que o consumidor armazene as informações por algum tempo viável para ele. Utilizando este parâmetro Ordem de Destino (`Destionation Order`), as mensagens armazenadas no Histórico podem ser organizadas de acordo com seu *timestamp* de publicação ou recebimento. O consumidor ainda pode especificar o tempo de validade da informação com o parâmetro Vida Útil (`Lifespan`). Através deste parâmetro, o consumidor define quando as mensagens devem ser retiradas do Histórico. Finalmente, Vivacidade (`Liveness`) permite que produtores enviem aos consumidores a situação de seus serviços, indicando se eles continuam ativos. O consumidor deve indicar se deseja receber este alerta.

Dados com qualidade são de grande importância em aplicações no domínio da saúde. Dados errôneos, incertos e ausentes não podem ser permitidos, pois as decisões tomadas com base em dados de baixa qualidade podem até comprometer a vida dos pacientes envolvidos. Há um grande desafio com relação à qualidade dos dados coletados de pacientes por meio de sensores e dispositivos vestíveis, pois eles podem ser ausentes e imprecisos (HICKS et al., 2019).

Os fatores que impactam de forma negativa na qualidade dos dados que são coletados estão relacionados principalmente ao dispositivo, a rede e ao usuário (CHO et al., 2021). Os fatores relacionados aos dispositivos (DÜKING et al., 2018; KARKOUCH et al., 2016) são: mau funcionamento de *hardware*, degradação ao longo do tempo, limite de espaço e qualidade do dispositivo. A vida útil limitada da bateria também é um fator crucial, pois esses dispositivos coletam dados de forma contínua. Logo, podem descarregar e interromper a coleta de dados. Já os fatores relacionados a rede resultam em problemas de

conexão (HARDY et al., 2018), mais exatamente como a perda de sinal e desconexões. Os fatorem também são relacionados ao usuário (REINERMAN-JONES; HARRIS; WATSON, 2017), por exemplo, a não utilização dos dispositivos vestíveis podem implicar na ausência de dados importantes para inferir o seu estado de saúde. Além da não utilização dos dispositivos, o mau uso também é um fator importante que impacta na qualidade dos dados. Com isso, os fatores citados afetam diretamente na exatidão e integridade dos dados. Garantir a qualidade e a confiabilidade do contexto em redes de sensores que coletam dados fisiológicos de pessoas é um desafio, pois elas são propícias a falhas tanto na aquisição, processamento e entrega dos dados (HUZOOREE; KHEDO; JOONAS, 2019).

2.2 Processamento de Eventos Complexos

Eventos são dados que representam algo que tenha ocorrido no mundo real ou em um sistema de software. Eles são constituídos de um *timestamp* (hora em que o evento foi gerado) e um *payload* (carga de dados). Em uma aplicação de fenotipagem digital podemos ter um evento *Location* que representa a posição geográfica do indivíduo onde possui a hora que a informação foi gerada e as informações de latitude e longitude que contemplam a carga útil do evento. Já um fluxo de eventos é definido como um conjunto de eventos gerados em um determinado intervalo de tempo.

O Processamento de Eventos Complexos (CEP) é um paradigma de programação concebido para analisar e reagir a fluxos de eventos em um tempo próximo ao real por meio de consultas contínuas (RAHMANI; BABAEI; SOURI, 2021). Diferente das abordagens de análise de dados tradicionais, onde os dados são armazenados e depois analisados, no CEP a análise ocorre à medida que os eventos são gerados.

Para analisar esse fluxo de eventos, o CEP fornece algumas primitivas. Elas são classificadas em primitivas de padrão e primitivas de transformação (CUGOLA; MARGARA, 2012). As primitivas de padrão (conjunção, disjunção, repetição, negação e sequência) são utilizadas para a detecção de padrões (e.g. gerar um evento complexo quando um evento E_1 ocorrer n vezes, ou, quando o evento E_1 ocorrer em conjunto com E_2). Já as primitivas de transformação (filtrar, dividir, compor, projetar, agregar e enriquecer) permitem filtrar, modificar ou correlacionar os eventos (e.g., filtrar um conjunto de eventos que obedeçam determinadas condições).

Cada consulta contínua é executada por um *Event Processing Agent* (EPA). Um EPA tem como função analisar, reagir, manipular e gerar eventos derivados (eventos complexos) (CUGOLA; MARGARA, 2012). Esses eventos complexos são destinados a consumidores ou a outro EPA. Os EPAs instanciam consultas contínuas escritas a partir de uma linguagem de processamento de eventos. Uma linguagem bastante conhecida e utilizada é a *Event Processing Language* (EPL) da *engine* de inferência CEP Esper

(EsperTech, 2022). Ela define operadores e primitivas para a análise desses fluxos de eventos, e sua estrutura tem similaridades com a *Structured Query Language* (SQL). Ao interconectar várias EPAs, constrói-se uma *Event Processing Network* (EPN). A Figura 1 apresenta o modelo geral de uma EPN.

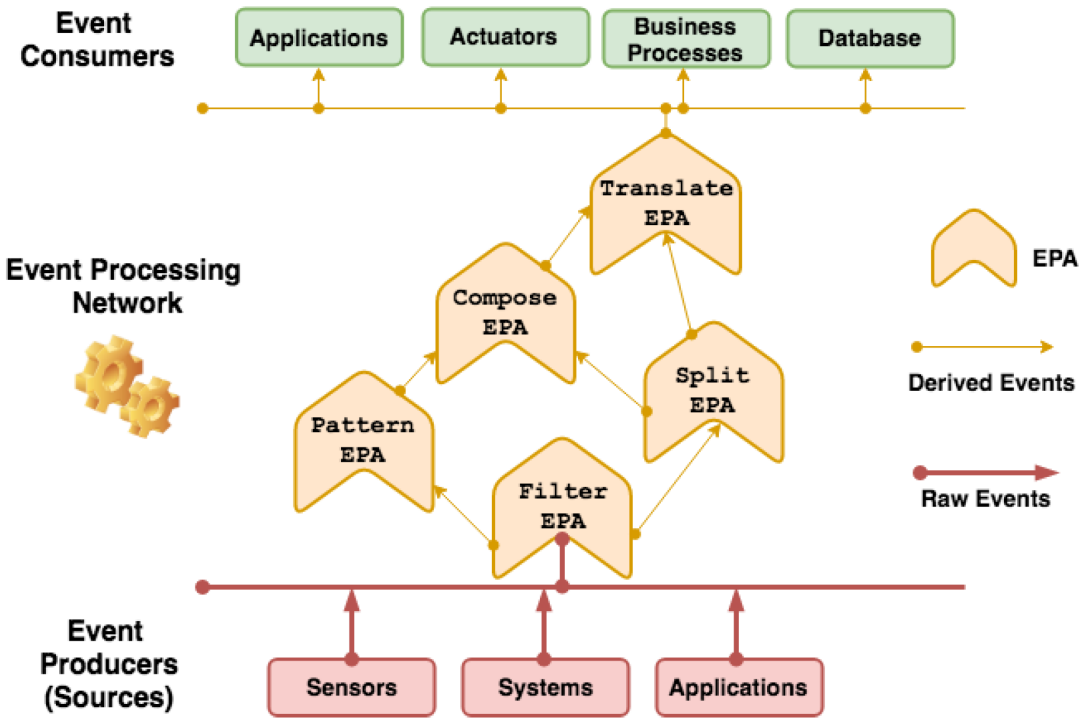


Figura 1 – Modelo geral de uma EPN (BEZERRA et al., 2021).

Para processar esse fluxo de eventos, o CEP utiliza o conceito de janelas de tempo. As janelas de tempo são partições de contexto que subdivide os eventos em intervalos de tempo considerando seu *timestamp* (EsperTech, 2022). Ela pode, por exemplo, conter eventos *Location* de um indivíduo dos últimos 30 segundos para serem analisados. As janelas de tempo são classificadas em lotes (*landmark/batch*) e deslizantes (*sliding*). As janelas *landmark* realizam o processamento considerando apenas os eventos naquele lote. Não consideram eventos adjacentes que estejam em outros lotes. Já as janelas *sliding* deslizam com relação aos eventos recebidos para considerar eventos adjacentes.

2.3 Middleware M-Hub/CDDL

O M-Hub/CDDL é um *middleware* concebido para facilitar o desenvolvimento de aplicações de Internet das Coisas (IoT) com requisitos de QoC. O *middleware* é composto por duas camadas, são elas: o *Mobile Hub* (M-Hub) e a *Context Data Distribution Layer* (CDDL). O M-Hub executa em dispositivos móveis Android para aquisição de dados de contexto do *smartphone* e de dispositivos pessoais inteligentes que possuem tecnologias como *Bluetooth Classic* (BT) e *Bluetooth Low Energy* (BLE) (SILVA et al., 2020). Já o

CDDL é responsável pelo processamento e distribuição dos dados de contexto obtidos por meio do M-Hub e pode ser executado tanto em dispositivos Android, aplicações *desktop* e em servidores na nuvem (GOMES et al., 2017). Ela fornece aos desenvolvedores de aplicações móveis, que utilizam dados de sensores do *smartphone* e de dispositivos vestíveis, mecanismos para garantir requisitos de QoC. A Figura 2 apresenta a utilização da infraestrutura do M-Hub/CDDL em um cenário de *Ambient Assisted Living* (AAL) para monitoramento de pacientes.

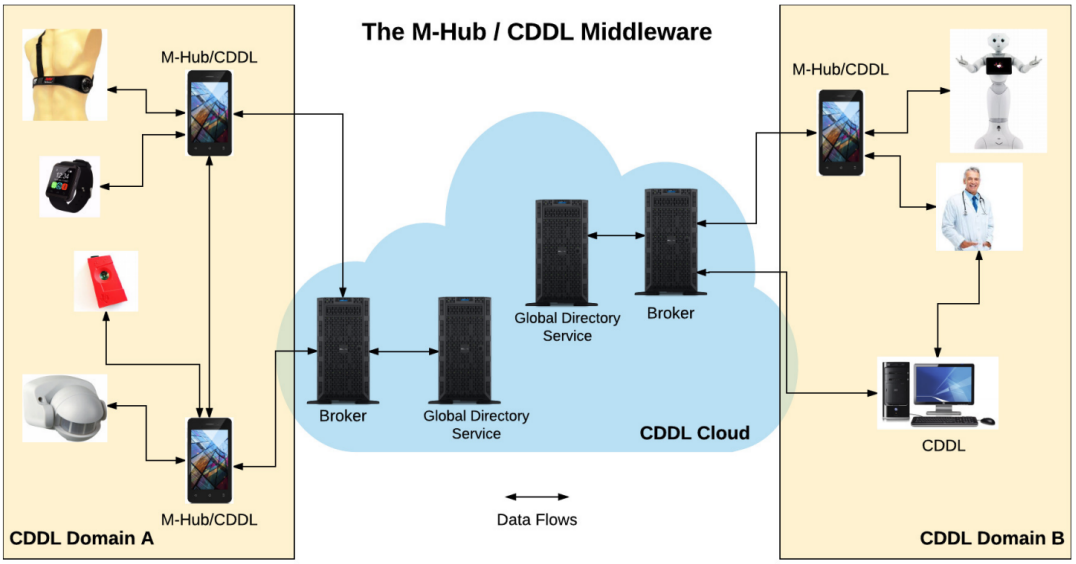


Figura 2 – Visão geral do M-Hub/CDDL em um cenário de AAL (GOMES et al., 2017).

O M-Hub é responsável pela descoberta e interação com dispositivos inteligentes encontrados na proximidade do *smartphone* por meio do *Bluetooth*. Quando o dispositivo é encontrado, o M-Hub realiza a busca do *driver* responsável por realizar a interação com o dispositivo para obtenção dos dados. Os dados coletados podem ser enriquecidos com metadados de QoI (e.g., acurácia, confiança, tempo de medição, localização). Quando as informações são oriundas de sensores, alguns metadados podem ser fornecidos pelos fabricantes, como no caso da acurácia e do tempo de medição. O M-Hub ainda dispõe de mecanismos para processar e reagir ao fluxo de eventos gerado por esses dispositivos. Os dados coletados por meio do M-Hub são enviados ao CDDL por um barramento de eventos (*EventBus*).

O CDDL ao receber os dados do M-Hub, computa dinamicamente os valores dos parâmetros de QoI suportados que não são fornecidos na camada de aquisição. Além de computar parâmetros de QoI, o CDDL permite ao desenvolvedor especificar o grau QoS ao realizar a distribuição das informações de contexto. As informações são distribuídas por um protocolo leve para sensores e pequenos dispositivos. O MQTT é um protocolo para distribuição de mensagens de sensores baseados no modelo *pub/sub* muito utilizado no contexto de IoT (MISHRA; KERTESZ, 2020). O CDDL fornece métodos publicadores

e subscritores para envio e recebimento das informações de contexto, respectivamente. Ele prover métodos de conexão que permite tanto a conexão com uma infraestrutura de *Broker MQTT* interna, que intermedeia a comunicação local entre os publicadores e os subscritores, mas também com uma infraestrutura remota. As informações de contexto são disponibilizadas para os consumidores em forma de serviços. Os serviços no CDDL são considerados como fontes provedoras de contexto disponíveis (e.g., sensor de frequência cardíaca de uma *smartband*). Os consumidores se inscrevem em serviços de interesse. Ao inscrever-se o desenvolvedor pode definir filtros para receber apenas as informações que possuem um determinado grau de qualidade. Pode ainda definir monitores para gerar notificações quando uma informação de contexto não atende aos parâmetros de QoI especificados.

Vale enfatizar que tanto os filtros como os monitores instanciam regras CEP para reagir sobre o fluxo de eventos, e toda lógica e ação a ser utilizada é de responsabilidade do desenvolvedor. Como essas regras são concebidas pelo desenvolvedor da aplicação, ele deve possuir conhecimentos desse paradigma de programação, além disso, tenha conhecimentos necessários para configurar os parâmetros de QoC.

2.4 Linguagens de Modelagem de Domínio Específico

As *Domain Specific Language* (DSL) são abordagens da Engenharia Dirigida a Modelos (MDE) que objetivam reduzir a complexidade do desenvolvimento de *software* por meio de modelos. No contexto da MDE, modelos são definidos como um conjunto de elementos que descrevem alguma realidade física, abstrata e hipotética (MELLOR et al., 2004). Na MDE os modelos são considerados artefatos, e por isso são definidos com base em outros modelos. Os modelos que descrevem outros modelos são conhecidos como metamodelos. Brambilla, Cabot e Wimmer (2017) definem metamodelo como uma linguagem de modelagem que fornece maneiras de descrever a classe de modelos que podem ser representados por essa linguagem.

Uma DSL é uma linguagem focada em um domínio específico. Ela é formada por uma sintaxe abstrata, por uma ou mais sintaxes concretas e que satisfaçam uma determinada semântica (HAREL; RUMPE, 2000). A sintaxe abstrata corresponde a um metamodelo que possui conceitos, abstrações e relações do domínio da aplicação. É necessário que os conceitos do metamodelo estejam próximos do domínio da aplicação para serem facilmente compreendidos pelos usuários. A sintaxe abstrata possui ainda uma semântica estrutural que representa as regras de relacionamento entre seus elementos. A sintaxe concreta representa uma notação (e.g., gráfica, textual) que objetiva facilitar a usabilidade da linguagem. Para especificar os requisitos de QoC das aplicações de fenotipagem digital foi desenvolvido um metamodelo que considera as características em

comum dessas aplicações.

2.5 Transformação entre Modelos

A transformação entre modelos consiste na geração de modelos por meio de um processo de conversão de um modelo de origem para um modelo de destino (KLEPPE; WARMER; BAST, 2003). Esse processo pode ser realizado de forma automática, semi-automática ou manual. Na transformação automática não há a necessidade da intervenção humana durante o processo; na semi-automática o desenvolvedor escolhe quais os elementos do modelo vão ser utilizados na transformação, e por fim, na transformação manual o desenvolvedor é que realiza a geração do artefato de *software*. Como um dos objetivos da transformação entre modelos é automatizar o processo de desenvolvimento de *software*, por meio da geração de artefatos (e.g., código-fonte, documentação), é recomendado que esse processo aconteça de forma automática (SILVA, 2015; ARNDT, 2016).

Um mecanismo de transformação é formado por um conjunto de regras que descrevem como um modelo na linguagem de origem pode ser convertido em um modelo na linguagem de destino (PARREIRAS, 2012). Parreiras (2012) define ainda que uma regra de transformação representa um ou mais elementos da linguagem de origem que pode ser transformado em um ou mais elementos da linguagem de destino. Para realizar a transformação entre modelos de forma automática é necessário utilizar uma DSL (KLEPPE; WARMER; BAST, 2003). A mais conhecida é a *Atlas Transformation Language (ATL)* criada pela comunidade Eclipse e introduzida no *framework Eclipse Modeling Framework (EMF)* para prover as definições de transformação. Com isso, é possível definir um metamodelo que representa o domínio específico de uma aplicação.

As transformações podem ser classificadas de acordo com seus resultados, logo, os tipos de transformações são: *Model-to-Model (M2M)*, *Model-to-Code (M2C)* ou *Model-to-Text (M2T)* (ARNDT, 2016). Na M2M o artefato gerado é outro modelo. Já na M2C são gerados códigos-fontes. Por fim, na M2C além de códigos fontes são gerados também artefatos textuais que podem representar casos de testes, documentações, entre outros. Para o mecanismo de transformação concebido neste estudo, utilizamos o tipo M2C, pois a partir do processo de transformação são gerados os código-fonte específicos do M-Hub/CDDL.

Os geradores de código podem ser baseados em três tipos de abordagens, são elas: *Framework-Specific Modeling Languages (FSML)*, padrões de códigos e *templates* (KOCH, 2006). A abordagem FSML é composta por uma sintaxe abstrata e um mapeamento entre essa sintaxe e a *Application Programing Interface (API)* de um *framework-base*. A abordagem de padrões de código utiliza uma técnica de interpretação lógica. Por fim, na abordagem baseada em *templates* são utilizados arquivos de texto padronizados e

instrumentados com códigos que realizam consultas em um modelo especificado. A Figura 3 apresenta como é realizada a transformação entre modelos na abordagem baseada em *templates*. Para o mecanismo de transformação implementado neste estudo, escolhemos a abordagem baseada em *template* por ser de fácil implementação e por ser a mais utilizada.

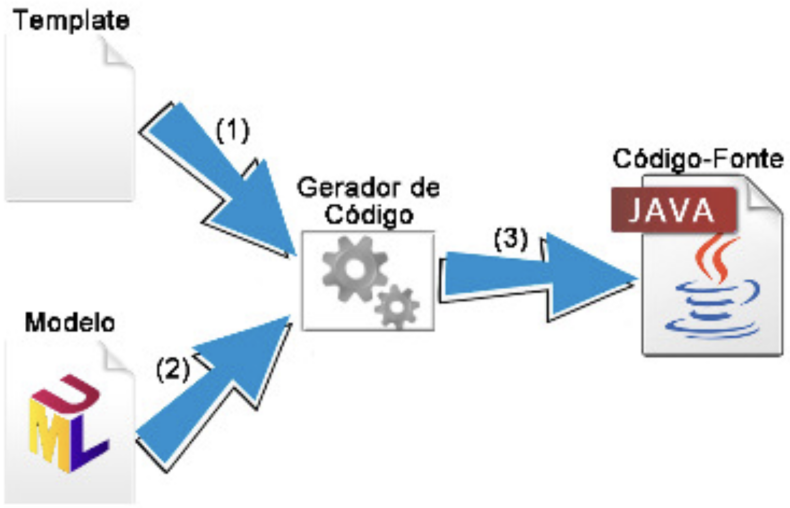


Figura 3 – Geração de código baseada em *templates* (ARNDT, 2016).

3 Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos que conceberam plataformas de *software* para apoiar estudos de fenotipagem digital. Os trabalhos foram selecionados por meio da Revisão Sistemática da Literatura (RSL) realizada por Mendes et al. (2022a), na qual os autores identificaram e caracterizaram as aplicações de sensoriamento e conjuntos de dados públicos para fenotipagem digital de uma perspectiva técnica. Para seleção dos estudos foram definidos os seguintes critérios, são eles: (i) concebem plataformas de *software* que apoiam os desenvolvedores na concepção de aplicações de fenotipagem digital; (ii) realizam aquisição de informações de contexto oriundas de sensores do *smartphone* ou de dispositivos vestíveis; (iii) disseminam as informações de contexto coletadas para aplicações consumidoras. A análise dos estudos selecionados objetivou verificar se as propostas abordavam algum processo ou metodologia para auxiliar desenvolvedores na incorporação de requisitos de QoC em aplicações para fenotipagem digital desenvolvidas por meio das plataformas identificadas.

3.1 Funf

O *Funf* (AHARONY et al., 2011) fornece um conjunto de funcionalidades que permite a coleta e distribuição das informações de contexto. O *Funf* possui um componente intitulado *Funf Manager* responsável por selecionar os sensores que serão utilizados para coletar as informações de contexto e configurar a frequência de coleta dos dados com a finalidade de minimizar o consumo de bateria. A possibilidade de alterar a taxa de amostragem dos dados é relevante quando considerado as limitações de bateria dos dispositivos e o consumo de banda das aplicações. A solução ainda conta com um *buffer* onde os dados são armazenados temporariamente, e a cada três horas, são enviados para um servidor na nuvem. Poder armazenar temporariamente informações em um *buffer* também é um requisito de QoS, pois acessar constantemente a rede pode consumir elevar o consumo de energia do dispositivo.

3.2 Purple Robot

O Purple Robot (SCHUELLER et al., 2014) é uma *framework* que apoia a criação de aplicações móveis, que coletam dados de sensores, por uma ferramenta de autoria *web* que auxiliam pesquisadores que não possuem conhecimento em desenvolvimento de *software* a desenvolverem suas próprias aplicações de aquisição de dados dos sensores do *smarphone* e de dispositivos vestíveis para à realização dos seus estudos.

Para avaliar a solução proposta foi desenvolvido um aplicativo intitulado *Mobilyze*. A aplicação visa realizar intervenções para reduzir os sintomas depressivo aumentando o envolvimento do usuário em atividades. O sistema coleta dados de sensores do *smartphone* para identificar os estados de saúde do usuário que podem ser relevantes para o tratamento, como localização, atividade, contexto social e humor. Como tal, o Mobilyze consiste em três componentes principais: (1) um aplicativo voltado para o paciente que inclui lições e ferramentas específicas do aplicativo, (2) um aplicativo para *smartphone* que coleta dados do sensor e fornece interações telefônicas que suportam a detecção de contexto funções e (3) uma estrutura do lado do servidor para coletar dados de usuário não estruturados e estruturados.

3.3 AWARE

O *AWARE* (FERREIRA; KOSTAKOS; DEY, 2015) é uma plataforma de software Android, reutilizável, para pesquisas em computação móvel sensível ao contexto. O foco principal está na aquisição, distribuição e inferência de contexto. A plataforma disponibiliza uma biblioteca onde permite ao desenvolvedor pesquisador conceber sua aplicação. A Figura 4 apresenta a arquitetura da plataforma.

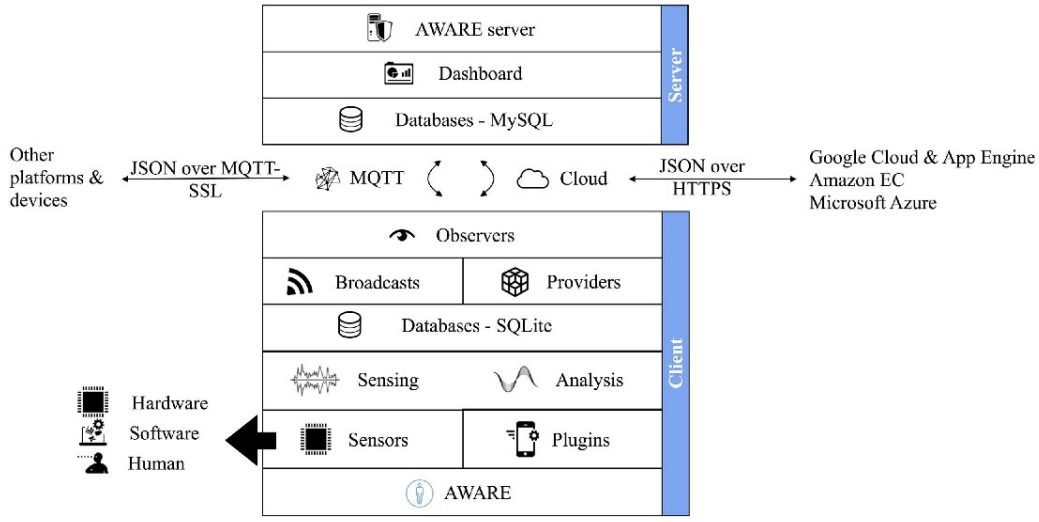


Figura 4 – Arquitetura da plataforma Aware (FERREIRA; KOSTAKOS; DEY, 2015).

A sua arquitetura é composta pelas camadas *Aware Client* e *Aware Server*. A camada *Aware Client* é responsável pela aquisição das informações de contexto oriundas dos sensores físicos e virtuais. Após a coleta dessas informações, nessa mesma camada as informações são processadas e inferidas situações de alto nível. As informações de alto nível são processadas por meio dos *plugins*. Cada *plugin* é responsável por alguma situação de interesse (e.g., sociabilidade, mobilidade, atividade física, sono). Já a camada *Aware Server* possui recursos de banco de dados na nuvem e *dashboards* para visualização

das informações. As informações apresentadas em tempo real são enviadas por meio do protocolo MQTT. O estudo aborda algumas medidas para minimizar a perda de dados. A primeira delas é utilizar os requisitos de QoS do MQTT, e a segunda é evitar que os processos de coleta de dados sejam interrompidos pelo Android. Para avaliar a solução proposta, foram desenvolvidos alguns estudos de caso, e com base neles foram avaliados aspectos de consumo de energia e escalabilidade.

3.4 Sesus

O *Sesus* é uma ferramenta que consiste em duas aplicações móveis (XIONG et al., 2016). A primeira delas é responsável por coletar os dados de sensores do *smartphone* e de dispositivos vestíveis utilizados pelo indivíduo monitorado. Os dados coletados são processados por um servidor na nuvem. A outra aplicação é utilizada pelo profissional de saúde para gerenciar o estudo. O estudo é gerenciado por um protocolo idealizado pelo profissional. O protocolo contêm as informações necessárias para aplicar o estudo, e os dados de sensores que serão coletados do *smartphone* do indivíduo. A Figura 5 apresenta a arquitetura da plataforma *Sensus*.

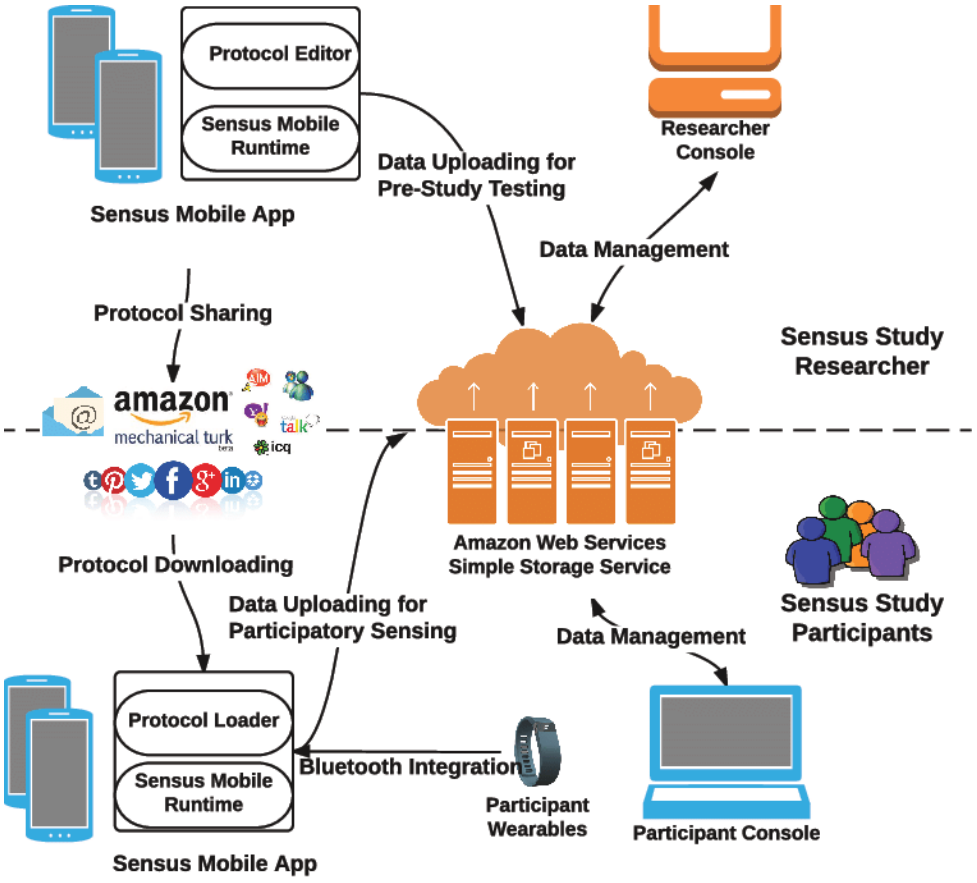


Figura 5 – Arquitetura da plataforma *Sensus* (XIONG et al., 2016).

Foi desenvolvido um estudo de caso com uma equipe de psicólogos clínicos. Eles desenvolveram um protocolo para coleta de dados no qual o foco era a ansiedade social em uma população de estudantes universitários. Com esse experimento, os autores observaram que os psicólogos foram capazes de projetar protocolos *Sensus* eficazes sem treinamento em linguagens de programação e desenvolvimento móvel.

3.5 Beiwe

O *Beiwe* (TOROUS et al., 2016) é uma plataforma que possui os seguintes componentes: Uma aplicação *web* permite que os pesquisadores especifiquem o conteúdo do aplicativo, quais sensores são usados para coleta de dados e como eles são amostrados. Uma aplicação móvel responsável pela coleta e distribuição de dados de diversos sensores do *smartphone*. Por fim, o *Beiwe* integra-se com os serviços em nuvem da *Amazon Web Services (AWS)* para processar os dados coletados. A Figura 6 apresenta o fluxo de trabalho da plataforma *Beiwe* para realizar o processo de fenotipagem digital.

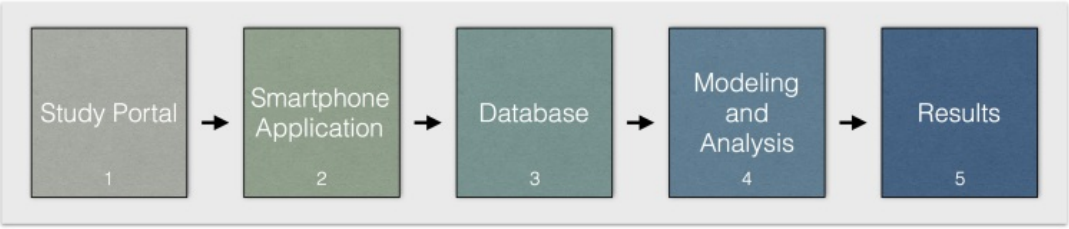


Figura 6 – Fluxo para fenotipagem digital da plataforma *Beiwe* (TOROUS et al., 2016).

Os dados coletados por meio da aplicação móvel são enviados para um servidor na nuvem quando há uma conexão de rede *Wi-Fi*, portanto, os dados são armazenados em um *buffer* no dispositivo até a realização do envio. Armazenar as informações em *buffer* é um dos requisitos de QoS importante, pois em determinados momentos a aplicação pode não possuir conexão com a internet, logo, essas informações são armazenadas para serem enviadas posteriormente, ao estabelecer uma conexão.

3.6 SituMan

O *SituMan* (TELES et al., 2017) é um serviço reutilizável desenvolvido para o sistema operacional que identifica situações da rotina do indivíduo com base em dados de sensores do *smartphone*. A inferência da situação é utilizada para solicitar autorrelatos em momentos oportunos quanto para que os profissionais de saúde mental conheçam seu cotidiano. A Figura 9 apresenta a arquitetura do *SituMan*.

A inferência das situações é realizado pelo módulo *Situation Inference Engine* com bases nos dados que são providos pelo *Context Provider*. O módulo *Context Provider*

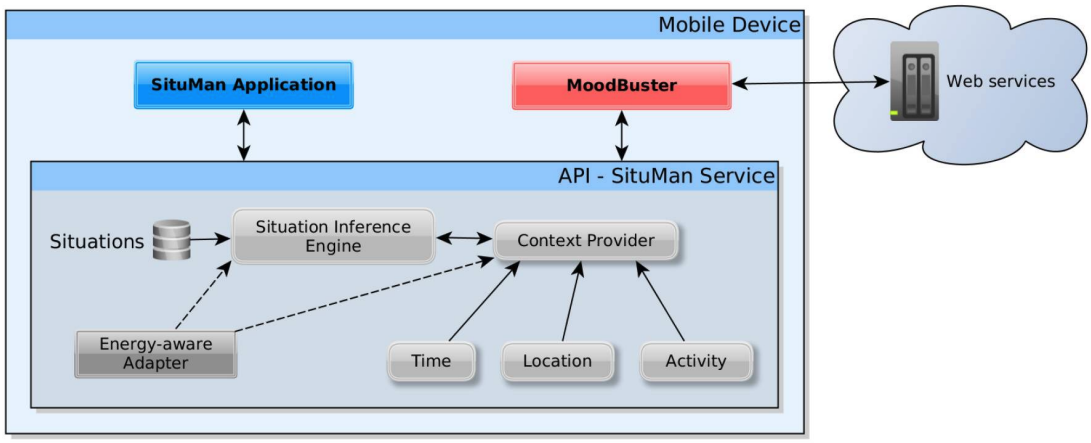


Figura 7 – Arquitetura do *SituMan* (TELES et al., 2017).

permite a coleta de dados relacionados a localização e atividade que o indivíduo está realizando. Os autores reconhecem que os dados de sensores utilizados podem em alguns momentos gerar dados imprecisos, como é o caso GPS utilizado pelos autores no estudo e a *Activity Recognition API* do *Google*. Diante dessas questões, a solução proposta pelos autores não aborda o provimento de mecanismos de QoC para garantir a qualidade das informações que são utilizadas pelo módulo *Situation Inference Engine*. Ao utilizar uma informação imprecisa, ela pode não representar o real contexto do indivíduo.

Para avaliação do *SituMan*, foram realizados dos experimentos. O primeiro deles avaliou a satisfação do usuário com as abordagens para definir e identificar situações de interesse. Já o segundo experimento foi realizado para avaliar a precisão do mecanismo de inferência.

3.7 *Lamp*

O *Lamp* (WISNIEWSKI; HENSON; TOROUS, 2019) é uma aplicação que realiza testes de cognição neuropsicológico por meio de jogos e questionários. Para isso são utilizados dados coletados de forma ativa (e.g., por questionários) e passiva (e.g., sensores do *smartphone* e dispositivos vestíveis). A Figura 8 apresenta a sua arquitetura. Ela é composta por quatro componentes, são eles: (i) aplicação móvel; (ii) o *Dashboard*; (iii) banco de dados; (iv) e o *Cortex*.

A aplicação móvel é utilizada para realizar a interação do usuário com os casos de teste. Por meio dela são coletados dados de questionários e de sensores do *smartphone* quando especificados, tais como acelerômetro, localização e pedômetro. Também são coletados dados referente ao uso da aplicação, por exemplo, quanto tempo uma pessoa leva para responder um questionário. O *Dashboard* é utilizado por profissionais de saúde que permite criar, personalizar e agendar questionários. Além disso, é possível acessar em

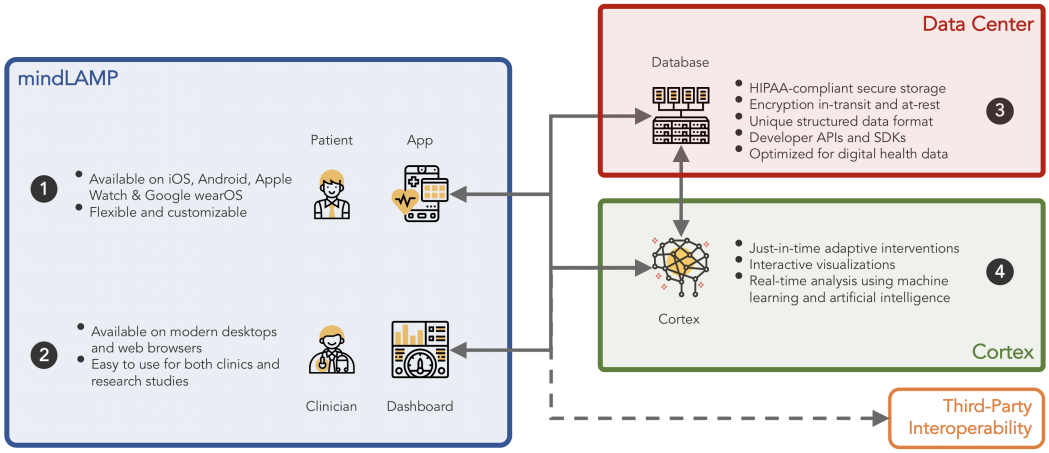


Figura 8 – Arquitetura da plataforma *Lamp* (WISNIEWSKI; HENSON; TOROUS, 2019).

tempo real as informações sobre os usuários. Já o *Cortex* foi projetado para recuperar as informações do banco de dados e extrair informações sobre aspectos comportamentais e de saúde dos usuários.

3.8 *OpenDP*

O *OpenDP* (MENDES et al., 2022b) é um *framework open source* desenvolvido para o sistema operacional Android que objetiva facilitar o desenvolvimento de aplicações móveis para fenotipagem digital. Ele é responsável por gerenciar a coletar dados brutos de sensores físicos e virtuais do *smartphone*, permitir a incorporação de módulos de inferência de situações (e.g., mobilidade, sociabilidade), e por fim, compõe fenótipos digitais a partir de informações geradas pelos módulos e processamento. A Figura 9 apresenta a arquitetura do *framework OpenDP*.

O *OpenDP* é composto por duas partes, são elas: o *core* e o *plugin*. O *core* é a camada onde está centralizado o controle dos recursos como: gerenciamento de sensores, gerenciamento dos módulos de processamento e dados brutos, gerenciamento de *plugins*, e a composição de fenótipos. Os fenótipos são distribuídos para uma infraestrutura de *Broker* no servidor por meio de uma instância do CDDL. Como dito anteriormente, o CDDL possui mecanismos para incorporar requisitos de QoC das aplicações, mas o suporte a QoC não foi abordado pelos autores no desenvolvimento do *OpenDP*. Por fim, o *plugin* é a camada responsável por adicionar novos módulos de processamento de informações de contexto.

Para validação da solução proposta foram desenvolvidos dois estudos de casos. No primeiro deles foi concebida uma aplicação intitulada *Sociability Monitoring Mobile System (SMMS)* cuja finalidade é identificar a socialização de usuários por meio da coleta de dados

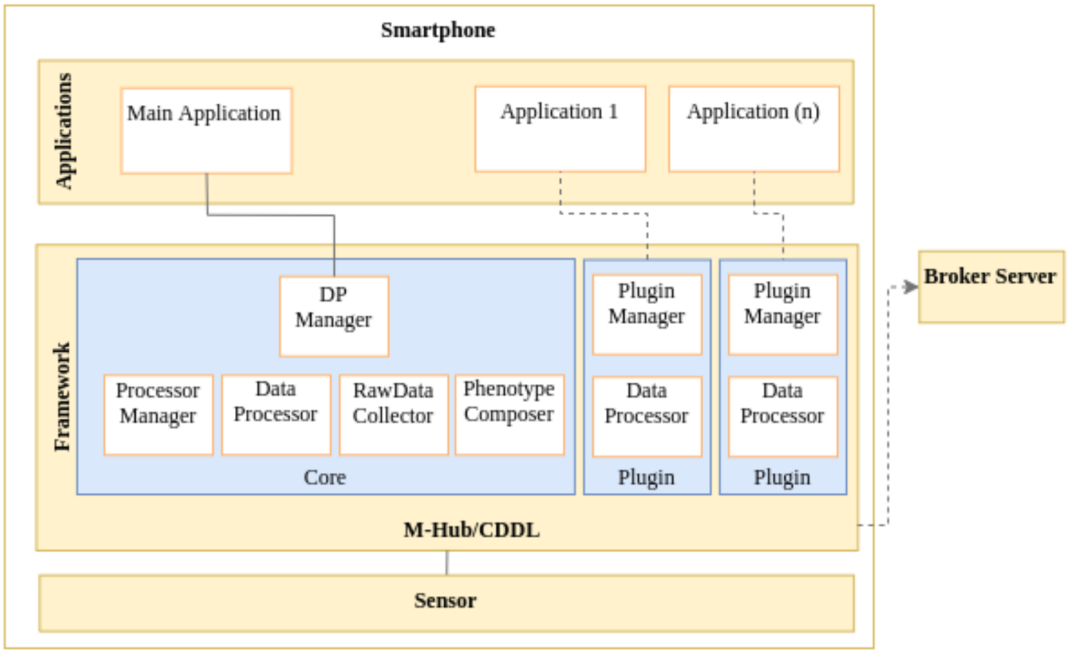


Figura 9 – Arquitetura da plataforma *OpenDP* (MENDES et al., 2022b).

de chamadas telefônicas, troca de mensagem de texto e conversação identificadas através d microfone do *smartphone*. Já no segundo estudo de caso foi adicionado a aplicação *SMMS* um *plugin* visando identificar o estado físico da usuária no decorrer do dia-a-dia. Com o estudo de caso, os autores realizaram uma avaliação experimental para analisar o impacto no consumo de energia do *smartphone*.

3.9 Análise Comparativa

As Tabelas 1 e 2 apresentam uma análise comparativa entre os trabalhos encontrados, objetivando apresentar como os estudos desenvolvidos abordam questões de aquisição e distribuição de informações de contexto, inferência e suporte a QoC. Os critérios adotados para realizar a comparação entre os trabalhos são:

- **Aquisição**: verifica se a plataforma possui mecanismos para a aquisição e distribuição de dados de contexto a partir de sensores.
- **Inferência**: verifica se a plataforma possui mecanismos para inferência de contexto de alto nível a partir dos dados de sensores coletados.
- **Especificação de requisitos**: verifica se a plataforma contempla mecanismos para a especificação de requisitos de QoC por parte das aplicações.
- **Suporte a QoC**: verifica se a plataforma possui suporte a mecanismos de QoI e QoS das informações de contexto.

- **Processo para incorporar QoC**: verifica se a plataforma dispõe de algum processo ou metodologia para auxiliar desenvolvedores a incorporar requisitos de QoC.
- **Monitoramento da QoC**: verifica se a plataforma contempla mecanismos para o monitoramento de parâmetros de QoC em tempo de execução da aplicação.

Tabela 1 – Comparação entre os trabalhos relacionados (Parte 1).

Nome (Referência)	**Aquisição**	**Inferência**	**Especificação de requisitos**
Funf (AHARONY et al., 2011)	Sim	Não	Sim
Purple Robot (SCHUELLER et al., 2014)	Sim	Não	Não
AWARE (FERREIRA; KOSTAKOS; DEY, 2015)	Sim	Sim	Não
Sesus (XIONG et al., 2016)	Sim	Sim	Não
Beiwe (TOROUS et al., 2016)	Sim	Não	Sim
SituMan (TELES et al., 2017)	Sim	Sim	Não
Lamp (WISNIEWSKI; HENSON; TOROUS, 2019)	Sim	Sim	Não
OpenDP (MENDES et al., 2022b)	Sim	Sim	Não
Nossa proposta	Sim	Não	Sim

Tabela 2 – Comparação entre os trabalhos relacionados (Parte 2).

Nome (Referência)	**Suporte a QoC**	**Processo para incorporar QoC**	**Monitoramento da QoC**
Funf (AHARONY et al., 2011)	Apenas QoS	Não	Não
Purple Robot (SCHUELLER et al., 2014)	Não	Não	Não
AWARE (FERREIRA; KOSTAKOS; DEY, 2015)	Apenas QoS	Não	Não
Sesus (XIONG et al., 2016)	Não	Não	Não

Beiwe (TOROUS et al., 2016)	Apenas QoS	Não	Não
SituMan (TELES et al., 2017)	Não	Não	Não
Lamp (WISNIEWSKI; HENSON; TOROUS, 2019)	Não	Não	Não
OpenDP (MENDES et al., 2022b)	Não	Não	Não
Nossa proposta	QoI e QoS	Sim	Sim

A aquisição dos dados é uma das principais etapas do processo de fenotipagem digital, logo todos os trabalhos apresentados nessa seção possuem esse mecanismo. As aplicações desenvolvidas nos estudos obtém as informações de contexto por meio de diversos sensores do *smartphone*, sejam eles físicos (e.g., GPS, acelerômetro, giroscópio) ou virtuais (e.g., chamadas telefônicas, tempo de tela ativado, aplicativos utilizados). Apenas (XIONG et al., 2016) e (MENDES et al., 2022b) possuem mecanismos que permitem a coleta de dados por meio de dispositivos vestíveis por meio da tecnologia *Bluetooth*. As informações de contexto são distribuídas em tempo real ou em lotes. Para distribuição em tempo real é utilizado o protocolo MQTT (FERREIRA; KOSTAKOS; DEY, 2015; MENDES et al., 2022a). Já para distribuição em lotes, os dados são enviados por meio do protocolo HTTP (AHARONY et al., 2011; FERREIRA; KOSTAKOS; DEY, 2015; XIONG et al., 2016; TOROUS et al., 2016; WISNIEWSKI; HENSON; TOROUS, 2019). Como (TELES et al., 2017) concebeu um serviço reutilizável para o sistema operacional, ele disponibiliza os dados por meio de uma API.

Informações de contexto de alto nível são geradas por meio de mecanismos de inferência de situações. Essas inferências podem ser realizadas tanto no *smartphone* como em servidores na nuvem. No *smartphone* são identificadas situações nas quais representam eventos comportamentais (e.g., atividade realizada, horário que o usuário socializou) (FERREIRA; KOSTAKOS; DEY, 2015; TELES et al., 2017; MENDES et al., 2022b). Já quando se trata de identificar padrões comportamentais, onde é necessário utilizar um conjunto maior de dados como entrada para modelos de aprendizagem de máquina, as inferências são realizadas em servidores na nuvem (XIONG et al., 2016; WISNIEWSKI; HENSON; TOROUS, 2019)

Com relação à especificação de requisitos de QoC, é possível alterar a taxa de amostragem dos dados obtidos por meio de sensores conforme o contexto da aplicação (AHARONY et al., 2011; TOROUS et al., 2016). Mesmo permitindo especificar a taxa de atualização com que as informações são geradas, as aplicações não permitem aos

desenvolvedores ou pesquisadores especificar de forma abrangente os requisitos de QoC necessários para o domínio da aplicação.

Para garantir o suporte a QoC, foram definidos parâmetros de QoS para garantir a entrega das informações de contexto (AHARONY et al., 2011; FERREIRA; KOSTAKOS; DEY, 2015; TOROUS et al., 2016), e variar a taxa de amostragem dos dados conforme o contexto da aplicação (AHARONY et al., 2011). Para situação no qual há uma perda de uma conexão entre o *smartphone* com o servidor é utilizado o parâmetro `History` para garantir que os dados serão salvos e enviados quando uma conexão com o servidor for estabelecida. Visando minimizar a perda de dados devido à perda de pacotes da rede utilizada para distribuição das informações, é definido o parâmetro `Reliability` para garantir que a informação será entregue exatamente uma vez. Já para alterar a taxa de atualização das informações foi definido o parâmetro `Refresh Rate`. Dentre os trabalhos encontrados que possuem suporte a QoC, nenhum deles permite monitorar o atendimento dos parâmetros de QoC definidos na aplicação.

Após análise dos trabalhos relacionados, observamos que as plataformas projetadas para adquirir e distribuir informações de contexto no âmbito da fenotipagem digital não atendem amplamente aos requisitos de QoC necessários para essas aplicações. O tratamento de QoC no domínio da fenotipagem digital ainda é incipiente na literatura, principalmente em plataformas de *middleware* voltadas para esta área. Portanto, esta pesquisa contribui para a literatura ao propor um processo que norteia a incorporação de requisitos de QoC em aplicações para fenotipagem digital. Além do processo, são concebidos mecanismos que facilitam a especificação de requisitos de QoC, considerando tanto aspectos de QoI quanto QoS, a incorporação desses requisitos e o monitoramento do atendimento dos parâmetros de QoC especificados.

4 Solução Proposta

A solução proposta consiste em um processo para incorporação de requisitos de QoC no desenvolvimento de aplicações para fenotipagem digital, e acompanhamento de sua execução. A Figura 10 apresenta as etapas deste processo. O processo consiste em cinco etapas, a saber: (i) especificação; (ii) transformação; (iii) incorporação; (iv) monitoramento; (v) visualização.

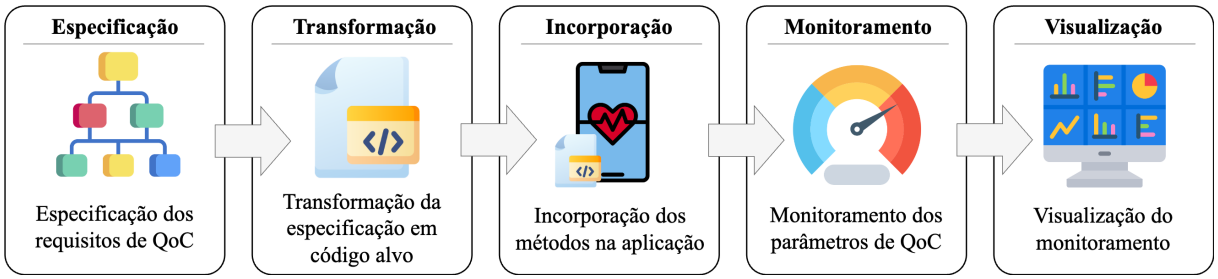


Figura 10 – Processo para incorporar requisitos de QoC em aplicações de fenotipagem digital.

A primeira etapa é especificar os requisitos de QoC da aplicação com a ajuda de um metamodelo. Após a formalização dos requisitos, a transformação para o código-alvo ocorre automaticamente por meio de um processo de transformação. Após a etapa de transformação, o desenvolvedor, de posse dos artefatos gerados com base no metamodelo, poderá utilizá-lo na implementação de sua aplicação, importando o código-fonte gerado para seu projeto. Em seguida, na etapa de monitoramento, são selecionadas as informações de contexto que atendem aos requisitos da aplicação que foram especificados, e calculadas algumas métricas para avaliar o nível de qualidade da aplicação. Por fim, na etapa de visualização, os resultados das métricas são apresentados por meio de um *Dashboard*, que exibe tanto dados em tempo real como dados históricos.

4.1 Metamodelo Proposto

Para o projeto do metamodelo, uma análise de domínio do problema foi realizada para identificar os conceitos, abstrações e relacionamentos entre as entidades. Esta análise de domínio produziu uma sintaxe abstrata que corresponde a um metamodelo. O metamodelo proposto foi arquitetado com base no padrão *Eclipse Modeling Framework*[1] (EMF). O EMF consiste em uma estrutura de modelagem baseada em um modelo de dados com recursos de geração de código. A Figura 11 apresenta o metamodelo proposto. Ele define

[1] https://www.eclipse.org/modeling/emf/

todos os conceitos identificados e seus respectivos relacionamentos. Os conceitos devem ser descritos para serem facilmente compreendidos pelo usuário. Inclui também as regras e restrições de relacionamentos entre classes de domínio.

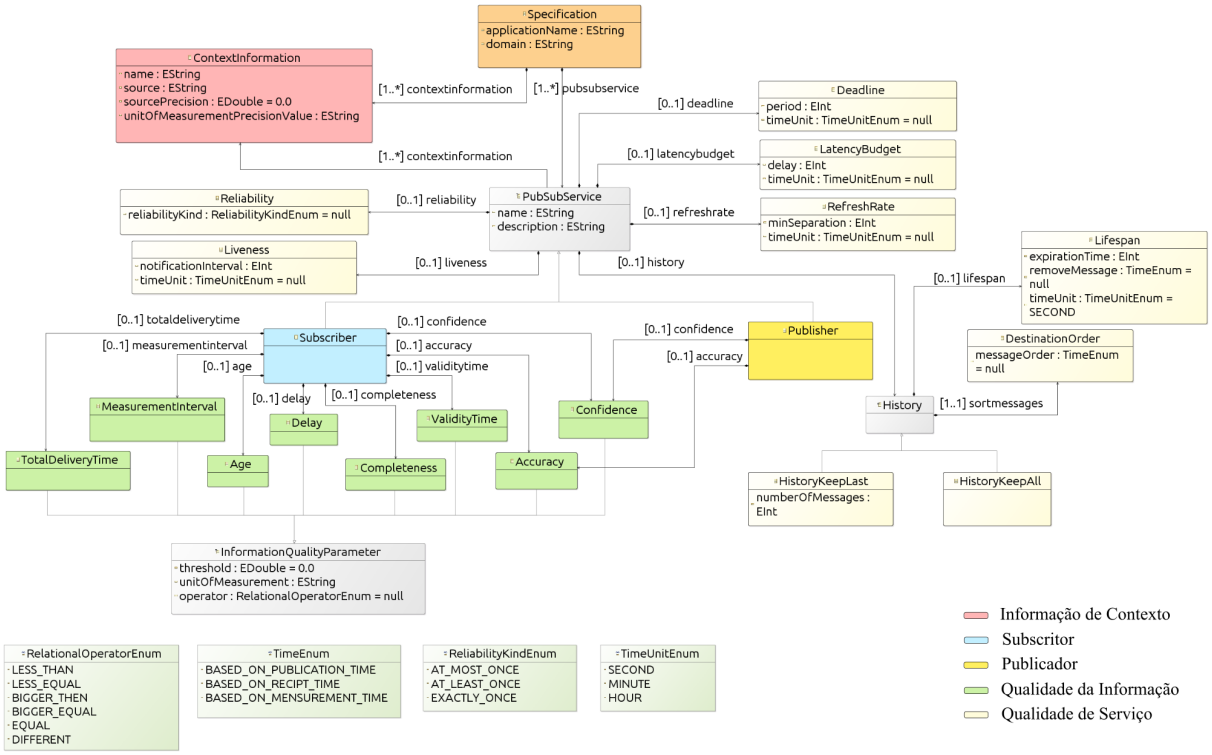


Figura 11 – Metamodelo proposto.

O metamodelo foi projetado com base no padrão pub/sub. Esse padrão consiste em definir produtores e consumidores de informações. Os produtores são responsáveis por disseminar informações para componentes de softwares ou aplicações que estão interessadas em receber essas informações. Essas informações são recebidas na aplicação por meio dos subscritores. Por exemplo, uma aplicação que monitora os batimentos cardíacos de um paciente, os dados de frequência cardíaca podem ser enviados, por meio de publicadores, para uma aplicação que executa no servidor. Logo, para receber essas informações, a aplicação que executa no servidor deve definir os subscritores.

Na fenotipagem digital, as aplicações consomem informações de contexto de, por exemplo, sensores. Além disso, essas aplicações distribuem essas informações para outras. O consumo e a disseminação de informações de contexto são definidos no metamodelo como `Publisher` (destacado na cor amarela) e `Subscriber` (destacado na cor azul). Cada produtor e consumidor de contexto está associado a uma ou mais informações de contexto (destacado na cor vermelha). Cada informação de contexto compreende um tipo específico de dados (e.g., frequência cardíaca), uma unidade de medida (e.g., bpm) e o valor de precisão da fonte provedora da informação.

Para cada `Publisher` e `Subscriber` é possível associar diferentes parâmetros de QoC. Os parâmetros de QoI, destacados na cor verde, são especificados para garantir que as informações recebidas ou enviadas tenham o nível de qualidade exigido pela aplicação. Ao definir um parâmetro de QoI, é necessário definir um valor limite, sua unidade de medida equivalente e um operador relacional. O operador relacional é utilizado para comparar o valor do parâmetro QoI contido na informação com o valor especificado. Para isso, as informações de contexto devem ser anotadas com a metainformação de QoI. A Tabela 3 apresenta os parâmetros de QoI definidos no metamodelo, e descreve as suas possíveis configurações.

Tabela 3 – Parâmetros de QoI e possíveis configurações.

Parâmetro	**Atributos**
`TotalDeliveryTime`	***threshold***: define o tempo máximo que a informação de contexto deve ser aceita pela aplicação. ***unitOfMeasurement***: define a unidade de tempo que representa o tempo total de entrega. ***operator***: define a operação utilizada para comparar o tempo total de entrega da informação com o *threshold* especificado. Exemplo: um dado de frequência cardíaca deve ser descartado caso seu tempo total de entrega seja maior que 10 segundos.
`MeasurementInterval`	***threshold***: define o intervalo de tempo em que os provedores de contexto devem gerar informações. ***unitOfMeasurement***: define a unidade de tempo que representa o intervalo de tempo entre as informações. ***operator***: define a operação utilizada para comparar o intervalo de tempo entre as informações geradas com o *threshold* especificado. Exemplo: um sensor de frequência cardíaca deve gerar novas leituras de dados a cada 1 segundo.

`Age`	***threshold***: define a idade máxima que a informação deve ter. ***unitOfMeasurement***: define a unidade de tempo que representa a idade da informação. ***operator***: define a operação utilizada para comparar a idade da informação com o *threshold* especificado. Exemplo: a aplicação só deve aceitar as informações que possuem idade menor ou igual a 10 segundos.
`Delay`	***threshold***: define o atraso máximo que a informação deve ter. ***unitOfMeasurement***: define a unidade de tempo que representa o atraso da informação. ***operator***: define a operação utilizada para comparar o atraso da informação com o *threshold* especificado. Exemplo: a aplicação só deve aceitar as informações que possuem um delay menor ou igual a 2 segundo.
`Completeness`	***threshold***: define o qual completa a informação deve ser. ***unitOfMeasurement***: define a unidade que representa o quão completa é a informação. ***operator***: define a operação utilizada para comparar o grau de completude da informação com o *threshold* especificado. Exemplo: uma medição de pressão arterial é composta por dois valores, o sistólico e o diastólico. A aplicação só deve aceitar a informação se ela possuir esses dois valores, ou seja, ter uma completude de 100%;
`ValidityTime`	***threshold***: define o prazo de validade da informação. ***unitOfMeasurement***: define a unidade de tempo que representa o prazo de validade da informação. ***operator***: define a operação utilizada para comparar o prazo de validade da informação com o *threshold* especificado. Exemplo: foi definido que o prazo de validade de uma informação de frequência cardíaca é de 10 segundos. Logo a aplicação deve descartar as informações que o prazo de validade expirou.

`Confidence`	***threshold***: define um valor limite para expressar se a informação de contexto possui um grau de confiança aceitável para o domínio da aplicação. ***unitOfMeasurement***: define a unidade de medida que representa o valor que indica o grau de confiança de uma informação de contexto. ***operator***: define a operação utilizada para comparar o grau de confiança da informação com o *threshold* especificado. Exemplo: a aplicação só deve receber as informações de atividade física que possui um grau de confiança igual ou maior que 70%.
`Accuracy`	***threshold***: define um valor limite para expressar se a informação de contexto possui um grau de acurácia aceitável para o domínio da aplicação. ***unitOfMeasurement***: define qual unidade de medida representa o valor que indica o grau de acurácia de uma informação de contexto. ***operator***: define a operação utilizada para comparar o grau de acurácia da informação com o *threshold* especificado. Exemplo: a aplicação só deve receber as informações de localização que possui uma acurácia de até 10 metros.

Os parâmetros de QoS, destacados na cor amarelo-claro, são especificados para garantir a qualidade do serviço de distribuição. Por exemplo, é possível definir o nível de confiabilidade da entrega de dados ao especificar o parâmetro `Reliability` e seu tipo. Também é possível definir o nível de QoS ao consumir as informações de contexto. Por exemplo, podemos definir um `Subscriber` para receber informações de um sensor de frequência cardíaca a cada 1 segundo, ao especificar o parâmetro `RefreshRate`. A Tabela 4 apresenta os parâmetros de QoS definidos no metamodelo, e descreve as suas possíveis configurações.

Tabela 4 – Parâmetros de QoS e possíveis configurações.

Parâmetro	**Atributos**

`Reliability`	***reliabilityKind***: Define o nível de garantia de entrega da informação. `AT_MOST_ONCE`: indica o melhor esforço, no qual o remetente não espera uma confirmação de entrega de mensagem. `AT_LEAST_ONCE`: indica que o remetente guarda uma cópia da mensagem até receber uma confirmação do destinatário de recebimento bem-sucedido. `EXACTLY_ONCE`: indica que é realizado um *handshake* entre o remetente e o destinatário para garantir que a informação será entregue exatamente uma vez. Exemplo: um dado de glicemia mensurado apenas algumas vezes ao dia deve ser realmente entregue ao servidor para ser persistido no banco de dados. Logo, um `Publisher` é definido com o parâmetro `Reliability` do tipo `EXACTLY_ONCE`.
`Liveness`	***notificationInterval***: valor que indica de quanto em quanto tempo um `Publisher` irá notificar os consumidores de contexto sobre sua situação, se está ativo ou não. O consumidor pode definir também de quanto em quanto tempo deseja receber essa notificação. ***timeUnit***: unidade de tempo que representa os intervalos de envio das notificações. Exemplo: um `Publisher` que envia informações de frequência cardíaca, pode enviar também a cada 10 segundos uma notificação que indica que continua ativo.
`Deadline`	***period***: indica o tempo máximo que o consumidor está disposto a esperar pela informação. ***timeUnit***: unidade de tempo que representa o período máximo que o consumidor está disposto a esperar pela informação. Exemplo: a aplicação deseja receber uma informação de frequência cardíaca, mas apenas se essa informação for recebida no máximo até 10 segundos do tempo que ela foi mensurada.

`LatencyBudget`	***delay***: valor que indica o período em que as informações de contexto são agrupadas para serem enviadas ou recebidas de uma única vez. ***timeUnit***: unidade de tempo que representa o período em que as informações de contexto são agrupadas. Exemplo: os dados de frequência cardíaca e ECG são agrupados por um período de 60 segundos para serem enviados de uma única vez.
`RefreshRate`	***minSeparation***: define um valor que indica a frequência com que as informações devem ser recebidas ou enviadas. ***timeUnit***: unidade de tempo que representa a frequência em que as informações são recebidas ou enviadas. Exemplo: um sensor de ECG gera dados a cada 500 milissegundo. Logo, a aplicação pode definir um `Subscriber` para receber esses dados apenas a cada 1 segundo.
`HistoryKeepLast`	***numberOfMessage***: quantidade máxima de mensagens que podem ser armazenadas no histórico. Exemplo: um `Subscriber` que envia dados de frequência cardíaca possui um histórico para armazenar até 300 mensagens. Pode ser utilizado para garantir a entrega dos dados em situações de conexão intermitente.
`DestinationOrder`	***messageOrder***: define a ordem que as informações são armazenadas no histórico. `BASED_ON_PUBLICATION_TIME`: indica que as informações são armazenadas com base no tempo de publicação. `BASED_ON_RECIPIT_TIME`: indica que as informações são armazenadas com base no tempo que foram recebidas. `BASED_ON_MEASUREMENT_TIME`: indica que as informações são armazenadas com base no tempo que foram mensuradas. Exemplo: as informações de frequência cardíaca são armazenadas no histórico com base no tempo que foram mensuradas.

`Lifespan`	***expirationTime***: indica o tempo de validade da informação deve permanecer no histórico. ***removeMessage***: indica o tempo que deve ser considerado para remover a mensagem do histórico. Se ela será removida considerando tempo de publicação, de recebimento ou o tempo que ela foi mensurada. ***timeUnit***: unidade de tempo que representa o tempo de validade. Exemplo: as informações recebidas há 5 minutos devem ser removidas do histórico.

Para exemplificar a utilização do metamodelo proposto em uma aplicação de fenotipagem digital, descrevemos o seguinte cenário hipotético: "Foi primeiramente realizado o levantamento de requisitos para o desenvolvimento de uma aplicação móvel utilizada para monitorar o usuário nos períodos em que ele realiza atividades físicas. Para isso, a aplicação necessita coletar dados de frequência cardíaca por meio de um dispositivo vestível conectado ao *smartphone*. Também é necessário coletar dados das atividades que o usuário está realizando no momento. As informações de atividade física são disponibilizadas por meio de uma API. Para um acompanhamento a longo prazo é necessário que esses dados coletados sejam persistidos em um servidor na nuvem. Para garantir um monitoramento com qualidade é necessário destacar alguns pontos, são eles: (i) as informações de atividade física são providas por uma fonte de dados que gera informações que possui um grau de confiança que varia de 0 a 100%; (ii) devido à mobilidade do usuário e a tecnologia de rede utilizada pode ocorrer em alguns momentos perda da conexão com o servidor, ocasionando perda de dados".

Diante do cenário descrito acima, observamos que é necessário definir subscritores e publicadores para receber as informações dos provedores de contexto e disseminar essas informações para o servidor, respectivamente. Portanto, podemos definir na aplicação dois `Subscribers`, um que recebe dados de frequência cardíaca, e o outro as informações de atividade física. E um `Publisher` que distribui essas informações para o servidor.

Como dito anteriormente, as informações de atividade física possuem um grau de confiança variável. Para garantir que as informações utilizadas pela aplicação represente com exatidão a atividade que o usuário está realizando no momento, no `Subscriber` de atividade física podemos definir o parâmetro `Confidence`, e determinar um *threshold* de 100%. Já para garantir a entrega das informações enviadas para o servidor mesmo quando

há perda de conexão, podemos definir um *buffer* no `Publisher` que acumula as últimas informações por um período, quando há uma conexão ativa com o servidor. Para isso definimos o parâmetro `HistoryKeepLast` com o *numberOfMessages* com valor 300.

4.2 Transformação e Incorporação dos Requisitos de QoC

Para incorporar os requisitos de QoC especificados é necessário utilizar plataformas de *middleware* que possuem os mecanismos necessários para implementar os parâmetros definidos no metamodelo proposto. Portanto, o processo de transformação do modelo em código-fonte específico do *middleware* inicia-se com a criação dos módulos geradores de código. A ferramenta Acceleo pode ser aplicada nesse processo, pois implementa o padrão MOF *Model to Text Language* (MTL) para transformar a especificação do modelo em código-alvo. O Acceleo é composto por duas principais estruturas, são elas: modelos e consultas. Os modelos têm um conjunto de instruções do Acceleo para gerar texto. As consultas são realizadas utilizando a *Acceleo Query Language* (AQL) e responsáveis por extrair as informações da especificação para serem utilizadas nos modelos. Após gerar os códigos-alvo referentes aos subscritores e produtores, o desenvolvedor pode incorporá-los a aplicação. A Figura 12 ilustra o processo de transformação.

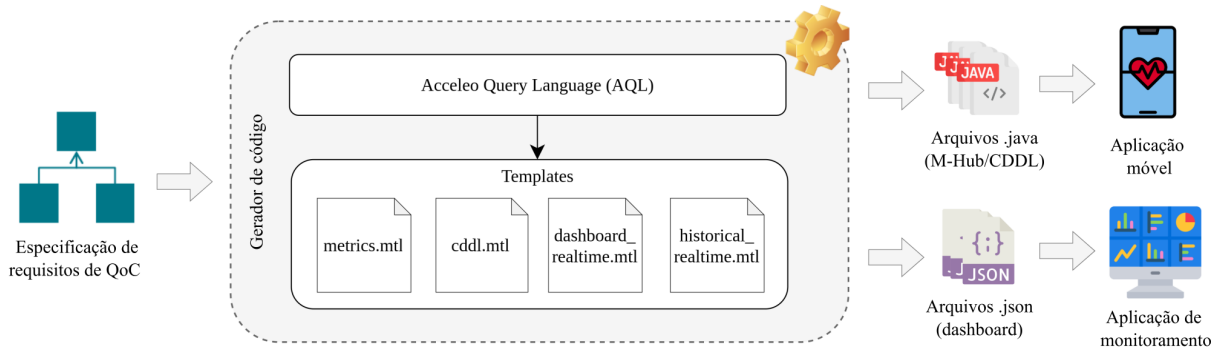


Figura 12 – Processo para geração de código alvo.

Os códigos alvos gerados pelo processo de transformação são classes em java específicas do *middleware* M-Hub/CDDL, pois ele possui os mecanismos necessários para implementar os requisitos que constam no metamodelo proposto. A etapa de transformação é necessária pelo fato de que esses requisitos de QoC no *middleware* devem ser explicitamente programados pelo desenvolvedor, no entanto, o processo de transformação automático simplificará esse esforço necessário. As classes, na linguagem java, geradas específicas do *middleware* são: `Subscriber.java`, `Publishers.java`. Elas são responsáveis por fornecer métodos para consumir e disseminar informações de contexto considerando os requisitos da aplicação que foram especificados.

Com relação ao acompanhamento do atendimento dos requisitos, o processo de transformação gera classes, na linguagem java, responsáveis por fornecer métodos para realizar o monitoramentos dos parâmetros especificados. As classes são: `Metrics.java`, `CollectorRegistryInstance.java` e `Gateway.java`. A classe `Metrics.java` possui um conjunto de métricas padrão definidas. Cada parâmetro de QoC é associado a um conjunto de métricas. As métricas definidas são registradas no aplicativo móvel por meio da classe `CollectorRegistryInstance.java`. Por fim, a classe `Gateway.java` é responsável por enviar os valores das métricas que foram computados para um sistema de armazenamento de métricas. O sistema utilizado na solução proposta é o Prometheus[2]. Ele coleta as métricas, em um tempo próximo ao real, e as persiste em um banco de dados de séries temporais.

Além dos códigos alvos gerados para serem incorporados na aplicação Android, o processo de transformação também gera os *Dashboards* utilizados para visualização e monitoramento das métricas dos parâmetros especificados. Os *Dashboards* são especificados no formato `json`, e importados para ferramenta de visualização. O Grafana[3] fornece uma interface composta por gráficos e tabelas para visualização interativa de dados. Os arquivos gerados para serem importados para o Grafana correspondem a *Dashboards* que apresentam tanto dados em tempo real, como também possibilita a consulta de dados históricos. Os arquivos são: `realtime_dashboard.json` e `historical_dashboard.json`.

4.3 Monitoramento dos Parâmetros de QoC

A etapa de monitoramento é realizada a partir do código-fonte gerado na etapa de transformação. Para avaliação das informações de contexto e monitoramento do atendimento dos requisitos especificados, utiliza-se CEP através de código Esper EPL. As regras CEP, descritas na linguagem Esper EPL, são geradas de forma automática por meio do mecanismo de transformação implementado, e incorporado automaticamente aos publicadores e subscritores instanciados pelo *middleware* M-Hub/CDDL. Para o monitoramento dos requisitos de QoC, foram definidas métricas objetivando mensurar o nível de qualidade da aplicação.

4.3.1 Métricas de QoI

Para os parâmetros de QoI foram definidas métricas para avaliar a qualidade das informações de contexto utilizadas pela aplicação. Portanto, para cada parâmetro de QoI presente no metamodelo proposto, definiram-se métricas que calculam a porcentagem de informações que atendem ao requisito especificado, e a média dos valores de cada

[2] https://prometheus.io/
[3] https://grafana.com/

informação de contexto referente a esse parâmetro. Com isso, podemos identificar o quão um provedor está gerando informações de contexto que não atendem aos requisitos da aplicação na qual essas informações serão utilizadas. Como as métricas são equivalentes para todos os parâmetros de QoI, vamos descrever como exemplo apenas as métricas para o parâmetro `Confidence`.

A métrica $confidence_{acceptedData}$ (Equação 4.1) calcula a porcentagem de informações que possuem uma acurácia aceitável pela aplicação. A equação referente a essa métrica é a seguinte:

$$confidence_{acceptedData} = \frac{TDConfidence * 100}{TD} \tag{4.1}$$

onde $TDConfidence$ representa o total de informações de contexto que possuem um valor de confiança aceitável pela aplicação, e TD o total informações recebidas ou enviadas.

Já a métrica $confidence_{avg}$ (Equação 4.2) calcula à confiança média das informações. A equação referente a essa métrica é a seguinte:

$$confidence_{avg} = \frac{\sum_{i=1}^{n} confidence(d_i)}{TD} \tag{4.2}$$

onde $confidence(d_i)$ representa grau de confiança de d_i, sendo que $d_i \in \theta = [d_1, \ldots, d_n]$ e d_n é a enésima informação de contexto.

4.3.2 Métrica de QoS

Para os parâmetros de QoS foram definidas métricas para avaliar a qualidade da entrega dos dados. Como o parâmetro o `History` visa definir que a aplicação armazene as informações de contexto por um período, foram definidas métricas para monitorar com relação à alocação de memória do dispositivo. A métrica $history_{capacity}$ (Equação 4.3) calcula a porcentagem da capacidade atual do histórico. A equação referente a essa métrica é a seguinte:

$$history_{capacity} = \frac{TDH * 100}{History_{tam}} \tag{4.3}$$

onde, TDH representa o total de mensagens que estão salvas no histórico e o $History_{tam}$ a capacidade máxima do histórico.

Já a métrica $history_{memory}$ (Equação 4.4) calcula o consumo de memória do histórico. A equação referente a essa métrica é a seguinte:

$$history_{memory} = \sum_{i=1}^{n} f_{bytes}(d_i) \tag{4.4}$$

onde $f_{bytes}(d_i)$ representa o valor em bytes de d_i, sendo que $d_i \in \theta = [d_1, \ldots, d_n]$ e d_n é a enésima informação de contexto do histórico.

Para o parâmetro `Lifespan` foi definida a métrica $data_{deleted}$ (Equação 4.5) que calcula a quantidade de informações deletadas do histórico da aplicação devido ao prazo de validade expirado. A equação referente a essa métrica é a seguinte:

$$data_{deleted} = \sum_{i=1}^{n} \begin{cases} 1, & \text{se } validytime(d_i) > validytime. \\ 0, & \text{caso contrário.} \end{cases} \tag{4.5}$$

sendo que $d_i \in \theta = [d_1, \ldots, d_n]$ e d_n é a enésima informação de contexto do histórico.

Para o parâmetro `Reliabilty` foi definida a métrica $reliability_{acceptedData}$ (Equação 4.6) que calcula a porcentagem das informações recebidas ou enviadas com o nível de garantia de entrega desejável.

$$reliabilty_{acceptedData} = \frac{TDReliability * 100}{TD} \tag{4.6}$$

onde $TDReliabilty$ representa o total de informações de contexto que possuem nível de garantia de entrega aceitável pela aplicação, e TD o total de informações recebidas ou enviadas.

Para o parâmetro `RefreshRate` foram definidas duas métricas. A primeira delas (Equação 4.7) calcula a porcentagem de informações que possuem uma taxa de atualização aceitável pela aplicação. A equação referente a essa métrica é a seguinte:

$$refreshRate_{acceptedData} = \frac{TDRefreshRate * 100}{TD} \tag{4.7}$$

onde $TDRefreshRate$ representa o total de informações de contexto que possuem uma taxa de atualização aceitável pela aplicação, e TD o total de informações recebidas ou enviadas.

Já a segunda (Equação 4.8) calcula a taxa de atualização média das informações. A equação referente a essa métrica é a seguinte:

$$refreshRate_{avg} = \frac{\sum_{i=1}^{n} refreshRate(d_i)}{TD} \tag{4.8}$$

onde $refreshRate(d_i)$ representa a taxa de atualização de d_i, sendo que $d_i \in \theta = [d_1, \ldots, d_n]$ e d_n é a enésima informação de contexto.

Para o parâmetro `Liveness` foram definidas três métricas, são elas: *downtime*, *uptime* e $uptime_{avg}$. A métrica *downtime* (Equação 4.9) calcula o período que o provedor de contexto ficou inativo, ela foi definida como:

$$downtime = \sum_{i=\Delta_t}^{n} t_i \tag{4.9}$$

onde Δ_t representa uma variação de tempo.

A métrica *uptime* (Equação 4.10) mensura a porcentagem do período que o provedor de contexto ficou ativo durante as 24 horas. A equação referente a essa métrica é a seguinte:

$$uptime = [\frac{24 - downtime}{24}] * 100 \tag{4.10}$$

Por fim, a métrica $uptime_{avg}$ (Equação 4.11) calcula o tempo médio de *uptime*, é definida como:

$$uptime_{avg} = \frac{\sum_{i=1}^{n} uptime_i}{n} \tag{4.11}$$

onde $uptime_i$ representa as porcentagens de *uptime* computadas, sendo que $uptime_i \in \theta = [uptime_1, \ldots, uptime_n]$ e $uptime_n$ é a enésima porcentagem de *uptime*.

Para o parâmetro `LatencyBudget` foram definidas as métricas: $latencyBudget_{acceptedData}$, $latencyBudget_{avg}$ e $compression_{percentagem}$. A métrica $latencyBudget_{acceptedData}$ (Equação 4.12) calcula a porcentagem dos grupos de informações que possuem um *delay* aceitável pela aplicação. A equação referente a essa métrica é a seguinte:

$$latencyBudget_{acceptedData} = \frac{TDLatency * 100}{TD} \tag{4.12}$$

onde $TDLatency$ representa o total de grupos que possuem um *delay* aceitável pela aplicação, e TD o total de grupos recebidos ou enviados.

A métrica $latencyBudget_{avg}$ (Equação 4.13) calcula o *delay* médio dos grupos de dados. A equação referente a essa métrica é a seguinte:

$$latencyBudget_{avg} = \frac{\sum_{i=1}^{n} delay(d_i)}{TD} \tag{4.13}$$

onde $delay(d_i)$ representa o *delay* de d_i, sendo que $d_i \in \theta = [d_1, \ldots, d_n]$ e d_n é a enésima informação de contexto.

A métrica $compression_{percentagem}$ (Equação 4.14) calcula a percentagem da redução do volume de dados que se obtém ao realizar a compressão de um conjunto de dados. A métrica $compression_{percentagem}$ é definida como:

$$compression_{percentagem} = \frac{f_{bytes}(f_{comp}(dataGroup_i)) * 100}{f_{bytes}(dataGroup_i)} \tag{4.14}$$

onde $f_{comp}(dataGroup_i)$ representa uma função que realiza a compressão do grupo de dados e $f_{bytes}(dataGroup_i)$ obtém seu tamanho em *bytes*.

4.3.3 Implementação do sistema de monitoramento

Como dito anteriormente, a avaliação e o monitoramento da QoC é realizada por meio de regras CEP geradas na etapa de transformação. Por exemplo, a regra descrita em 4.1 a seguir define que o `Subscriber ActivityDetectionSensor` receba apenas informações de atividade física que possuem um grau de confiança igual ou superior a 70%. As regras garantem que as informações utilizadas pela aplicação atendem aos requisitos especificados.

```
1 SELECT * FROM Message
2 WHERE (serviceName = 'ActivityDetectionSensor')
3 AND (confidence >= 70);
```

Listing 4.1 – Regra CEP para filtrar as informações de atividade física que possuem grau de confiança igual ou superior ao 70%.

Os valores das métricas também são calculados por meio de regras CEP. As regras descritas em 4.2 apresentam um exemplo de como as métricas de porcentagem e média são calculadas. As informações são armazenadas em uma janela deslizante de tamanho 60. Para calcular essas métricas é necessário ter a quantidade de informações de contexto que atendem ao requisito especificado, o total de informações recebidas na janela e os valores referente ao parâmetro especificado de cada informação de contexto.

```
1 INSERT INTO WinEventActivityDetectionSensor
2 SELECT confidence, COUNT(*) as total,
3 (SELECT COUNT(*)
4 FROM Message.WIN:LENGHT(60)
5 WHERE confidence >= 70) as accepted
6 FROM Message.WIN:LENGHT(60);
7
8 SELECT ((accepted * 100) / total) as percentage,
9 AVG(confidence) as average
```

```
10 FROM WinEventActivityDetectionSensor;
```

Listing 4.2 – Regras CEP para calcular as métricas a porcentagem de informações de atividade física que atenderam ao requisito de confiança especificado e a confiança média das informações.

Após mensurar as métricas, seus valores são enviados para o Prometheus por meio do Pushgateway[4]. O Pushgateway é responsável por disponibilizar um cache de métricas para realizar o intermédio entre o provedor das métricas (aplicação de monitoramento de saúde que executa no *smartphone*) e o Prometheus. A Figura 13 apresenta a arquitetura definida para o monitoramento das métricas definidas.

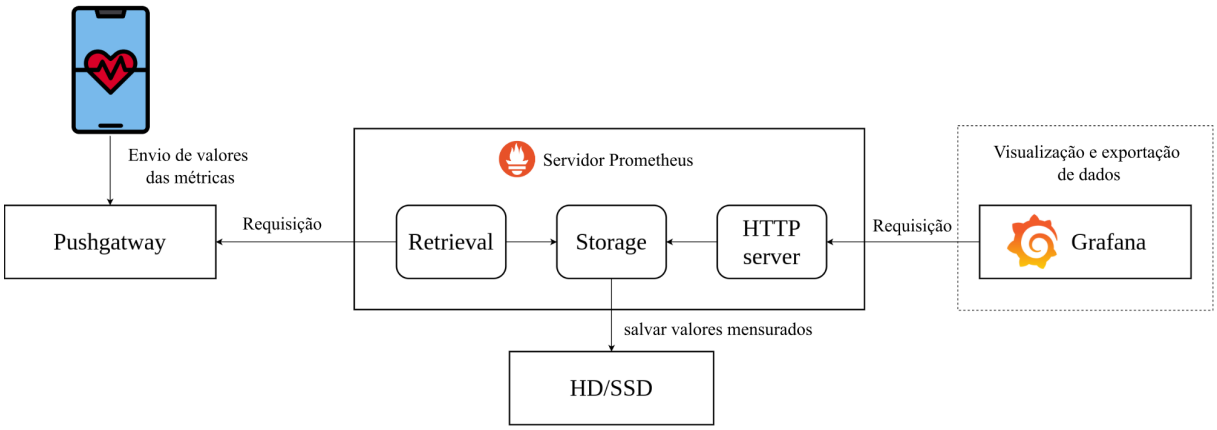


Figura 13 – Arquitetura para monitoramento da QoC da aplicação.

O Prometheus é composto basicamente por três serviços, são eles: Retrieval, Storage Service e Http Server. O Retrieval é responsável por realizar o envio dos valores das métricas para o Storage Service para serem salvas no banco. O Storage Service armazena essas métricas na memória ou em disco local, formando assim séries temporais. Por fim, o HTTP Server disponibiliza uma API Rest para aplicações de visualização de dados (e.g., Grafana) obter os dados e apresentá-los em um *Dashboard* para visualização e análise de dados.

4.4 Visualização

A utilização de uma ferramenta de *Dashboard* no contexto de monitoramento da QoC visa fornecer ao desenvolvedor um ambiente no qual seja possível acompanhar o nível de qualidade das instâncias da aplicação. Para isso foram disponibilizados dois *Dashboards*. Eles apresentam informações sobre cada `Publisher` e `Subscriber`, das métricas dos parâmetros especificado, e refletem o que foi definido na especificação de QoC para a aplicação. Para exemplificar, são apresentadas as informações das métricas para o *Subscriber*

[4] https://github.com/prometheus/pushgateway

que recebe dados de atividade física, no qual foram especificados os seguintes parâmetros, são eles: `History` e `Confidence`. O primeiro *Dashboard* apresenta informações das métricas em um tempo próximo ao real, como podemos ver na Figura 14. Já o segundo, ilustrado na Figura 15, apresenta dados históricos.

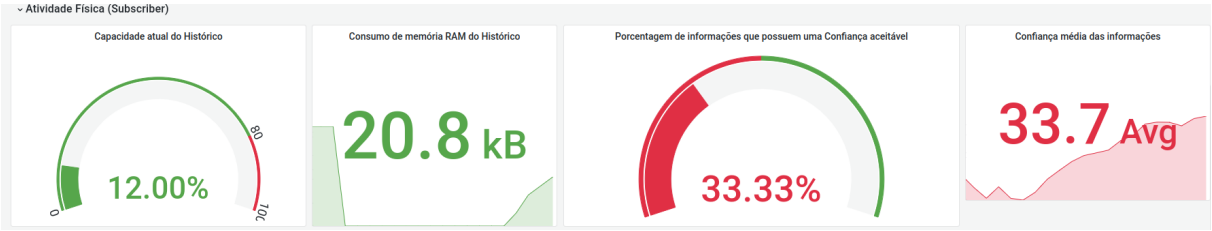


Figura 14 – *Dashboard* de tempo real.

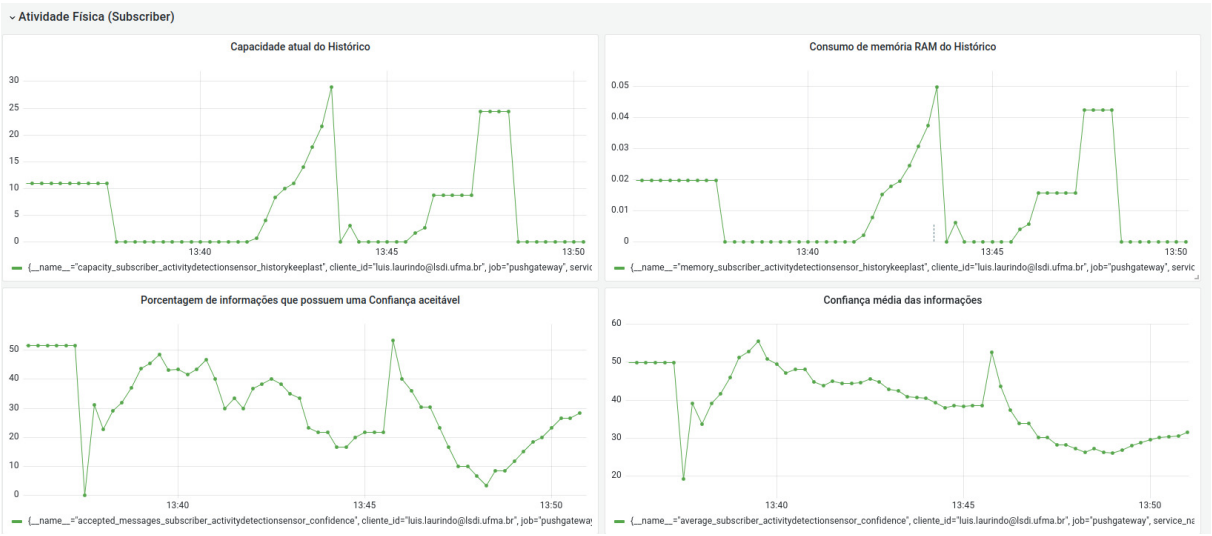


Figura 15 – *Dashboard* histórico.

No *Dashboard* de tempo real podemos observar o contexto atual da aplicação, tais como: a capacidade atual do histórico, o consumo de memória RAM das informações armazenadas no histórico, a porcentagem de informações que atendem ao requisito especificado para o parâmetro confiança e a confiança média das informações de atividade física. Já o *Dashboard* de dados históricos permite consultar dados de acordo com um intervalo de data especificado. Ter um ambiente no qual apresenta dados históricos é bastante útil por possibilitar ao desenvolvedor analisar o comportamento da aplicação com relação à qualidade ao longo do tempo em que a aplicação esteve em execução. Por exemplo, ao analisar as métricas referente ao histórico, percebemos que em determinados momentos a aplicação perdeu a conexão com o *Broker* MQTT, com isso as informações passaram a ser acumuladas no histórico até o restabelecimento da conexão. Ao restabelecer a conexão, as informações foram removidas do histórico e enviadas.

4.5 Considerações

Esse capítulo apresentou uma solução que consiste em um processo composto por etapas que permitam a especificação de requisitos, geração de código que implementam os mecanismos de QoC e a sua incorporação a aplicação e mecanismos para o monitoramento da QoC durante a execução da aplicação.

Os requisitos de QoC da aplicação são especificados por meio de um metamodelo baseado no padrão pub/sub. O metamodelo contempla uma ampla variedade de parâmetros de QoC que podem ser definidos tanto na aquisição como na distribuição dos dados. O desenvolvedor pode escolher e configurar os parâmetros conforme o contexto da aplicação.

Para incorporar esses requisitos na aplicação, foi desenvolvido um mecanismo de transformação para gerar os códigos-fonte específicos da plataforma de *middleware* M-Hub/CDDL que implementam os mecanismos de QoC.

Após incorporar os mecanismos de QoC na aplicação, é possível avaliar em tempo real quais informações de contexto atendem aos requisitos da aplicação por meio de regras CEP, e garantir a distribuição dos dados com o nível de qualidade necessário.

Com relação ao monitoramento foram definidas métricas para cada parâmetro de QoC. As métricas são computadas por meio de regras CEP geradas pelo mecanismo de transformação. Os valores das métricas são enviados para o Prometheus para serem persistidas no banco.

Por fim, são disponibilizados *Dashboards* que fornecem ao desenvolvedor um ambiente no qual permite acompanhar o nível de qualidade das instâncias da aplicação em tempo de execução, mas também permite acessar dados históricos, com base nas métricas definidas.

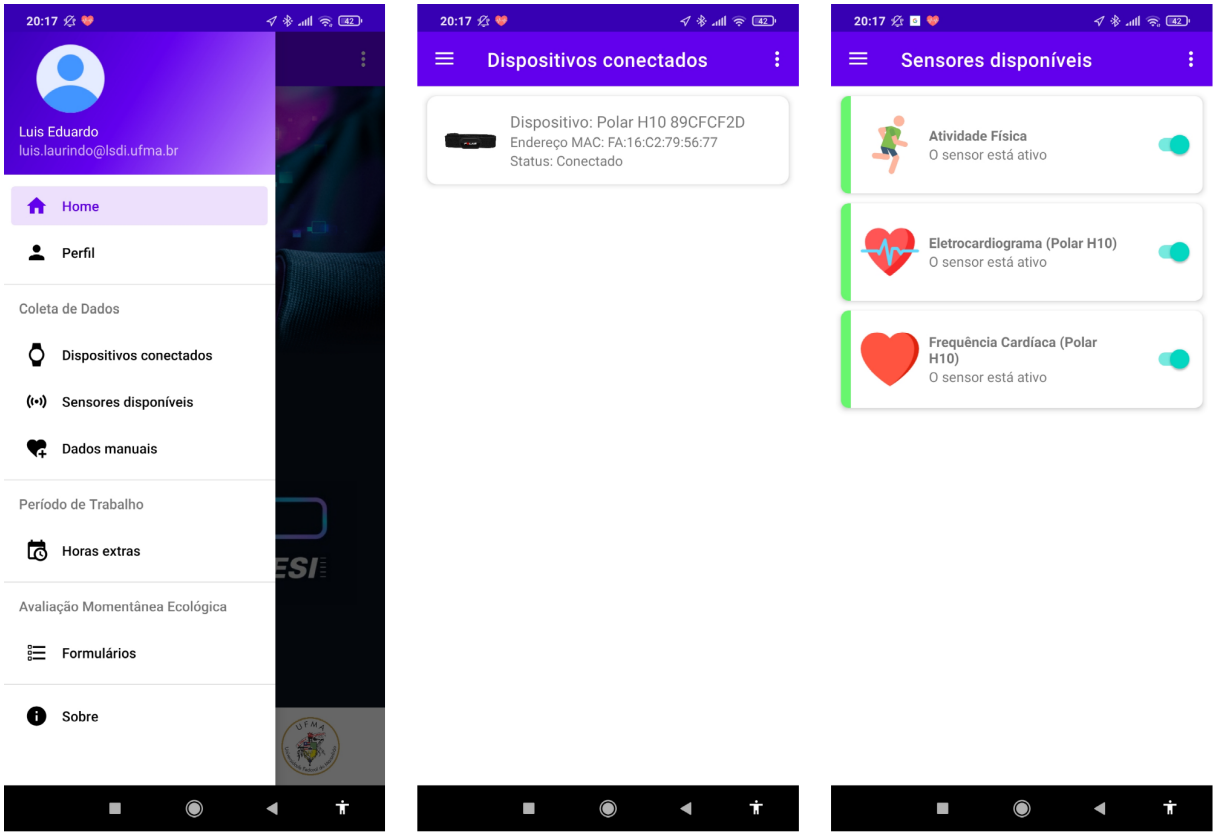
5 Estudo de Caso

Sistemas móveis de monitoramento de saúde são aplicações que executam em dispositivos móveis e visam inferir o estado de saúde dos usuários com base em informações oriundas principalmente de sensores. Essas aplicações utilizam tecnologias de comunicação sem fio para coletar dados de dispositivos vestíveis utilizados por usuários durante o período de monitoramento e distribuir essas informações. Como estudo de caso, foi utilizado um sistema desenvolvido pelo Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da Universidade Federal do Maranhão (UFMA) intitulado LSDi m-Health. O sistema é composto por um conjunto de aplicações, são elas: (i) uma aplicação móvel; (ii) um servidor em nuvem; (iii) e uma aplicação *Dashboard*. A aplicação móvel responsável pela aquisição e distribuição de dados foi implementada pelo discente autor dessa dissertação. O servidor foi implementado por um aluno de doutorado. Já a aplicação *Dashboard* foi desenvolvida por um aluno de mestrado.

A aplicação móvel Android se conecta a dispositivos vestíveis utilizando tecnologia de comunicação *Bluetooth*. Além de realizar a coleta dos dados, a aplicação móvel distribui essas informações por meio do protocolo MQTT para o servidor e para o *Dashboard*. Os requisitos de QoC foram incorporados na aplicação utilizando o processo proposto neste estudo e a plataforma de *Middleware* M-Hub/CDDL. O servidor é uma aplicação SpringBoot[1] que executa em um servidor em nuvem, responsável por receber as informações de contexto, inferir situações de interesse e armazenar essas informações. Por fim, o *Dashboard* é uma aplicação *web* desenvolvida na linguagem *Python* que apresenta aos profissionais da saúde os sinais vitais dos usuários monitorado por meio da visualização de dados em um tempo próximo ao real, e de dados históricos. A Figura 16 apresenta algumas telas da aplicação móvel Android. A primeira delas apresenta o menu da aplicação, em seguida são apresentados os dispositivos que estão conectados, e por fim, a última imagem mostra os sensores disponíveis no momento. Já à Figura 17 mostra o *Dashboard* apresentando os dados de frequência cardíaca em um tempo próximo ao real.

A aplicação móvel realiza uma busca para estabelecer uma conexão com os dispositivos vestíveis que estão próximo. Os dispositivo suportados pela aplicação são: Polar H10, Polar Verity Sense, Zephyr™ BioHarness 3.0 e Zephyr HxM. Ao estabelecer uma conexão, a aplicação encaminha um notificação para o servidor informando que um novo dispositivo foi conectado e os tipos de informações de contexto que aquele dispositivo fornece. O mesmo ocorre quando a conexão com o dispositivo é interrompida. Os dados são coletados em períodos específicos do dia que são definidos ao cadastrar o usuário no sistema. Os dados de Pressão Arterial, Glicose e SpO2 não foram possíveis ser obtidos de forma

[1] https://spring.io/projects/spring-boot



(a) (b) (c)

Figura 16 – (a) Tela que mostra o menu de opções. (b) Tela que mostra os dispositivos conectados. (c) Tela mostrando os sensores disponíveis e ativos.

Figura 17 – Dados de frequência cardíaca em tempo próximo ao real.

contínua, devido a não disponibilização de um *Software Development Kit* (SDK) para permitir a interação entre dispositivos que coletam esses tipos de dados e a aplicação móvel.

Logo, a aplicação permite aos usuarios inserir os dados de Pressão Arterial, Glicose e SpO2 manualmente. Com base nos sinais vitais coletados a aplicação realiza inferências para identificar alterações com base em uma faixa de valores definidos. Além dos dados de sinais vitais a aplicação coleta dados de atividade física fornecidos pela *Activity Recognition API*[2] do Google. Os dados fornecem a descrição do tipo de atividade que está sendo realizada (e.g., caminhando, correndo, parado) e o grau de confiança da informação. Visando garantir a entrega dos dados mesmo situações de conexão intermitente, a aplicação possui um mecanismo de *buffer* que permite armazenar os dados quando é identificado uma perca de conexão com o servidor. Por fim, para garantir a economia de recurso dos dispositivos a aplicação permite definir uma taxa de amostragem na aquisição dos dados e um período no qual os dados são agrupados para serem enviados de uma só vez.

5.1 Etapa de Especificação dos Requisitos de QoC

Seguindo o processo proposto nessa dissertação, a primeira etapa para a construção da aplicação no que se refere ao provimento de QoC foi a etapa de especificação de requisitos, conforme descrito a seguir.

Notificar as aplicações clientes sobre a situação da conexão do dispositivo vestível com o *smartphone*. A tecnologia *Bluetooth* possui uma distância limite para manter uma conexão ativa. No ambiente de monitoramento, em que o usuário monitorado pode se distanciar do *smartphone* que está conectado ao dispositivo, faz-se necessário notificar aplicações consumidoras sobre a situação do dispositivo, se está conectado ao *smartphone* ou não. Com isso, a aplicação móvel notifica o servidor sobre a situação da conexão entre o *smartphone* e os dispositivos vestíveis. Como a aplicação móvel publica informações de contexto de vários sensores (Frequência Cardíaca, ECG e Taxa de Respiração) foram definidos publicadores (`HeartRatePublisher`, `EcgPublisher`, `BreathingRatePublisher`) para cada tipo de informação de contexto, e foi especificado a eles o parâmetro `Liveness`. Esse parâmetro indica que os publicadores notificam o servidor, informando se o dispositivo vestível está conectado ao *smartphone* ou não. Foi definido que as notificações vão ser enviadas a cada 10 segundos.

Identificar as Atividades Físicas que o usuário realizou. A aplicação móvel identifica as atividades que os usuários estão realizando por meio da *Activity Recognition API*[3]. A API retorna todas as atividades com o grau de confiança que possui para cada uma delas, menos a que possui um valor igual a 0. Para garantir que a atividade fornecida realmente condiz com a atividade realizada, foi definido o parâmetro `Confidence` com um *threshold* de 70%. Com isso, a aplicação de monitoramento irá receber apenas as informações de atividade que tenham um grau de confiança a partir acima do *threshold*

[2] https://developers.google.com/location-context/activity-recognition
[3] https://developers.google.com/location-context/activity-recognition?hl=pt-br

especificado. Esse valor para o *threashold* foi definido com base em testes realizados com a API, onde foi possível observar que as informações acima de 70% realmente condizem com a atividade que o usuário estava realizando naquele momento.

O sistema deve garantir a entrega dos dados mesmo considerando conexões intermitentes que pode ser resultante da tecnologia de rede utilizada ou da mobilidade do usuário no período de monitoramento. Foi definido para cada *Subscriber* o parâmetro `HistoryKeepLast` para armazenar as últimas 300 informações nos períodos que não há falhas na conexão com a internet ou com o servidor. Ao definir o parâmetro `HistoryKeepLast` a aplicação ativa o mecanismo que persiste as informações em um *buffer*. Ao restabelecer a conexão, as informações armazenadas são enviadas para o servidor e persistida no banco de dados. Foi definido o parâmetro `Reliability` do tipo `EXACTLY_ONCE` para o `Publisher` que envia os dados de Pressão Arterial, Glicose e SpO2, já que a coleta ocorre apenas algumas vezes ao dia. Logo, faz-se necessário garantir a entrega dessas medições. Já para os publicadores que envia os dados obtidos de forma contínua (Frequência Cardíaca, ECG e Taxa de Respiração), foi definido o parâmetro `Reliability` do tipo `AT_MOST_ONCE`.

Disponibilizar mecanismos que permitam configurar a aplicação de forma a adequar o consumo de recursos de acordo com o dispositivo e tecnologia de rede utilizada para a transferência dos dados ao servidor. Foi especificado o parâmetro `LatencyBudget` para o `MessageGrupoPublisher`. Logo foi definido um `Delay` de 60 segundos para agrupamento e envio dos dados. Para minimizar o volume de dados que é gerado pelos sensores foi definido parâmetro `RefreshRate` para os `Subscribers` que recebem as informações de contexto dos dispositivos vestíveis. Para o parâmetro foi especificado o valor de 1 segundo que indica a taxa de amostragem dos dados.

Os requisitos acima descritos foram especificados de acordo com o metamodelo descrito na Seção 4. A especificação foi modelada utilizando a ferramenta Sirius[4]. Ela permite ao desenvolvedor definir uma especificação por meio de uma interface gráfica com base em um metamodelo definido. Na Figura 18, por limitações de espaço, optou-se por mostrar apenas um resumo representativo de como os requisitos foram especificados. Para o `HeartRateSubscriber` os dados de frequência cardíaca do Polar H10 devem ser recebidos a cada segundo e as informações armazenadas em um Histórico do tipo `HistoryKeepLast` que comporta as últimas 300 informações. Já o `HeartRatePublisher` é responsável por enviar as informações de contexto referente a frequência cardíaca e notificar as aplicações consumidoras sobre a situação da conexão do Polar H10 com o *smartphone*, se está ativa ou não. Por fim, o `ActivityDetectionSensorSubscriber` indica que os dados de atividade física são recebidos apenas se possuir um grau de confiança maior ou igual a 70%, além disso, foi definido um Histórico para armazenar as informações.

[4] https://www.eclipse.org/sirius/overview.html

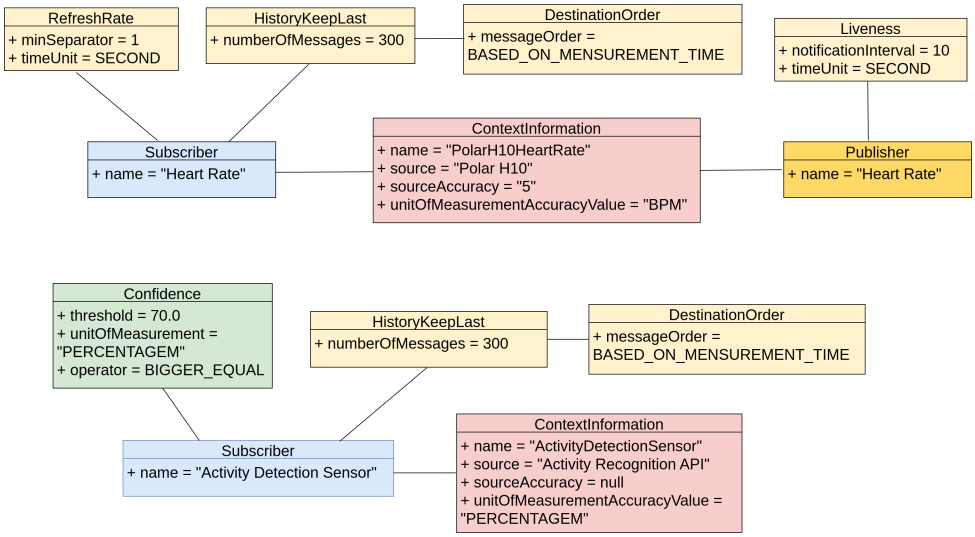


Figura 18 – Resumo da especificação dos requisitos de QoC.

5.2 Etapa de Transformação

Após a especificação de requisitos de QoC da aplicação realizou-se a etapa de transformação para então incorporar os requisitos na aplicação desenvolvida. A Figura 19 apresenta um exemplo de como foi realizada a especificação dos requisitos por meio da ferramenta Sirius. Em seguida o mecanismo de transformação foi executado, gerando os artefatos de código para serem incorporados.

Figura 19 – Especificação dos requisitos de QoC por meio da ferramenta Sirius.

Para aplicação móvel Android o processo de transformação gerou três classes, são elas: `Publishers.java`, `Subscribers.java` e `Metrics.java`. As classes `Subscribers.java`

e `Publishers.java` contêm métodos responsáveis por receber e distribuir informações de contexto, respectivamente. Já a classe `Metrics.java` possui atributos que representam as métricas e métodos para definição de seus valores. Para exemplificar, os trechos de códigos 5.1, 5.2 e 5.3 apresentam como foi gerado o método `createActivityDetectionSensorSub` que cria um subscritor para receber as informações de contexto de atividade física considerando os requisitos de garantia de entrega e confiança da informação.

Um dos parâmetros definidos na aplicação para o `Subscriber` de atividade física é o `History`, para garantir a entrega do dado mesmo em períodos de conexão intermitente. O trecho de código 5.1 apresenta como é especificado o parâmetro `History` no M-Hub/CDDL. A classe `HistoryQoS` é instanciada com um tipo, quantidade máxima de informações que podem ser persistidas e a ordem que elas vão ser armazenadas.

```
1  private static Subscriber createActivityDetectionSensorSub() {
2      Subscriber sub = SubscriberFactory.createSubscriber();
3      sub.addConnection(CDDL.getInstance().getConnection());
4
5      //... Oculta trechos de codigos anteriores
6
7      // History QoS
8      HistoryQoS history = new HistoryQoS();
9      history.setKind(HistoryQoS.KEEP_LAST);
10     history.setDepth(300);
11     DestinationOrderQoS order = new DestinationOrderQoS();
12     order.setKind(DestinationOrderQoS.BASED_ON_MENSUREMENT_TIME);
13     History history = new History(history, order);
14     sub.setHistory(history);
15
16     //...
17 }
```

Listing 5.1 – Definição do parâmetro `History` para o `Subscriber` que recebe dados de atividade física.

Com relação à confiança das informações, o trecho de código 5.2 apresenta como é gerado o método que garante que as informações recebidas atendam ao requisito de confiança especificado. As informações são selecionadas por meio de uma regra CEP que verifica se o valor de confiança da informação é maior ou igual a 70%. Caso essa condição seja satisfeita, as informações são enviadas para o servidor. Além disso, são computadas as métricas $history_{capacity}$ e $history_{memory}$ referente ao parâmetro `History`.

```
1  private static Subscriber createActivityDetectionSensorSub() {
2      //... Oculta trechos de codigos anteriores
3      sub.getMonitor().addRule("SELECT * FROM Message " +
```

```
                "WHERE (serviceName = 'ActivityDetectionSensor') " +
                "AND (confidence >= 70.0)", message -> {
        History history = activityDetectionSensorSub.getHistory();
        double capacity = capacityOfHistory(history);
        double memory = historyMemoryConsumption(history.readAll());

        Metrics.setHistoryCapacityActivityDetectionSensorSub(
            CDDL.getInstance().getConnection().getClientId(),
            "ActivityDetectionSensor",
            capacity
        );

        Metrics.setHistoryMemoryActivityDetectionSensorSub(
            CDDL.getInstance().getConnection().getClientId(),
            "ActivityDetectionSensor",
            memory
        );

        if(ConnectionBroker.getConnection().isConnected()) {
            List<Message> messages = history.takeAll();
            messages.forEach(message -> {
                sendMessage(message);
            });
        }
    });

    //...
}
```

Listing 5.2 – Seleciona apenas as informações que possuem um grau de confiança igual ou acima de 70%.

Já as métricas referente ao parâmetro `Confidence` são apresentadas no trecho de código 5.3 e seus valores são calculados por meio de regras CEP. A primeira regra cria uma janela na qual são armazenadas o valor de confiança de cada informação, a quantidade de informações que atenderam a regra dos 70% e o total de informações recebidas. Esses valores são utilizados para calcular a porcentagem de informações que atendem aos requisitos especificados ($confidence_{acceptedData}$) e a confiança média das informações recebidas ($confidence_{avg}$). Esses valores de porcentagem e média são obtidos por meio da segunda regra CEP descrita no exemplo.

```
private static Subscriber createSubscriberActivityDetectionSensor() {
    //... Oculta trechos de codigos anteriores

    sub.getRuleExecuter().addRule("INSERT INTO " +
        "WinEventActivityDetectionSensorConfidence " +
```

```
6           "SELECT confidence, count(*) as total, " +
7           "(SELECT count(*) FROM Message.win:length(60) " +
8           "WHERE confidence >= 70.0) as accepted",
9           (eventBeans, eventBeans1) -> {});
10
11      sub.getRuleExecuter().addRule(
12          "SELECT ((accepted * 100) / total) as percentage,
13          avg(confidence) as average " +
14          "FROM WinEventActivityDetectionSensorConfidence",
15          (e1, e2) -> {
16              MapEventBean eventMap = (MapEventBean) e1[0];
17              Double avg = (Double) eventMap.get("average");
18              Double percentage = (Double) eventMap.get("percentage");
19
20              Metrics.setActivitySubConfidenceAverage(
21                  CDDL.getInstance().getConnection().getClientId(),
22                  "ActivityDetectionSensor",
23                  avg
24              );
25
26              Metrics.setActivitySubConfidenceAcceptedData(
27                  CDDL.getInstance().getConnection().getClientId(),
28                  "ActivityDetectionSensor",
29                  percentage
30              );
31          });
32      //...
33  }
```

Listing 5.3 – Calculando as métricas por meio das regras CEP.

Para gerenciar as métricas por meio do Prometheu é necessário definir atributos que as representam. Esses atributos armazenam os valores, indicam qual deve ser o nome da métrica e quais rótulos ela deve ter. Os rótulos servem para realizar algum tipo filtragem, por exemplo, buscar apenas os valores da métrica *confidence_accepted_data* referente às informações de contexto de atividade física do usuário com o *cliente_id* paciente1. O *cliente_id* é utilizado para identificação do *smartphone* do usuário monitorado. O trecho de código 5.4 apresenta um exemplo da classe que contêm os atributos e métodos referente as métricas.

```
1  public class Metrics {
2
3      private static Gauge activitySubConfidenceAcceptedData =
4          Gauge.build()
5          .name("confidence_accepted_data")
6          .help("confidence_accepted_data")
```

```
7           .labelNames("cliente_id", "service_name")
8           .register(CollectorRegistry.getCollectorRegistry());
9
10      public static void setActivitySubConfidenceAcceptedData(
11          String clientId, String serviceName, Double value) {
12
13              activitySubConfidenceAcceptedData
14                  .labels(clientId, serviceName)
15                  .set(value);
16      }
17
18      // ..
19  }
```

Listing 5.4 – Atributo responsável por armazenar a porcentagem de infomações de atividade física que atenderam ao requisito especificado.

Os valores computados das métricas são enviados para o servidor do Prometheus por meio do Pushgateway. A classe responsável por enviar as informações da aplicação móvel para o Pushgateway também é gerada por meio do processo de transformação.

5.3 Etapa de Incorporação dos Requisitos

De posse dos artefatos de código gerados no processo de transformação foi realizada a importação para as aplicações. Primeiramente foram importadas na aplicação Android as classes `Publisher.java` e `Subscriber.java` que contêm os métodos para receber e distribuir as informações de contexto e calcular as métricas utilizadas para monitoramento da QoC. Em seguida, foram importadas as classes referente às definições das métricas. A classe `Metrics.java` possui atributos que representam as métricas e métodos para definição de seus valores. Ao definir um novo valor para uma métrica, o método `push` da classe `PushGateway.java` atualiza os valores no servidor de métricas para serem consumidos pelo serviço de monitoramento. Em seguida, após a importação das classes na aplicação Android, realizou a importação dos arquivos em JSON que contém as configurações necessárias para definir os *Dashboards* no Grafana.

5.4 Etapa de Monitoramento

Após a etapa de incorporação dos requisitos, nós realizamos o monitoramento de um usuário por um período de 4 horas para verificar se todos os componentes do sistema estavam funcionando corretamente. No decorrer do teste observamos que os mecanismos para garantir a QoC da aplicação funcionaram corretamente, e os valores das métricas estavam sendo enviados ao longo do período de monitoramento para o Prometheus,

e visualizadas em tempo real no *Dashboard*. A Figura 20 apresenta uma instância do *Dashboard* quando a aplicação estava em execução.

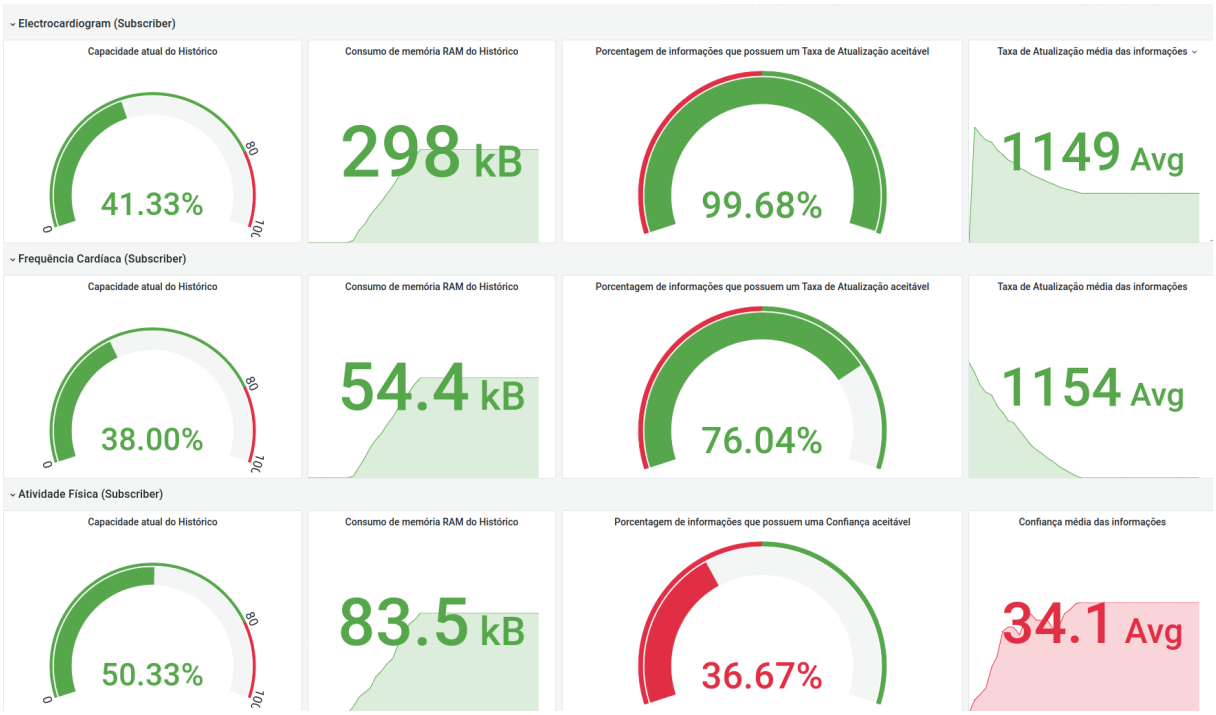


Figura 20 – Monitoramento em tempo real dos parâmetros de QoC.

Ao utilizar o monitoramento temos uma visão geral de como está o nível de QoC das instâncias da aplicação que estão sendo executadas no momento, no caso no monitoramento em tempo real. Ao observar a Figura 20 podemos identificar que nesse período do monitoramento a aplicação estava coletando dados do dispositivo Polar H10, mas houve uma eventual desconexão do *smartphone* com o servidor, pois as informações de frequência cardíaca e ECG estavam sendo acumuladas no histórico. Observamos isso por meio dos gráficos que apresentam a porcentagem de ocupação do histórico e o consumo de memória dele. Para as informações de atividade física observamos por meio do gráfico que apenas 36,67% possuíam uma confiança igual ou acima de 70%. Isso mostra que a *Activity Recognition API* infere muitas informações que possuem uma baixa confiança para o contexto da aplicação. Observamos isso por meio do gráfico que apresenta a média do grau de confiança com base em todas as informações de atividade física que foram geradas pela *Activity Recognition API*.

5.5 Considerações

O estudo de caso descrito objetivou avaliar o processo proposto para incorporar requisitos de QoC em aplicações de fenotipagem digital no âmbito da saúde. Para tanto, o

processo foi aplicado no desenvolvimento de uma aplicação móvel que coleta e distribui informações de contexto oriundas de sensores do *smartphone* e de dispositivos vestíveis.

Por meio do metamodelo definido foi possível auxiliar o desenvolvedor a expressar os requisitos de QoC da aplicação de forma não ambígua, a partir de um conjunto bem definido de parâmetros de QoC configuráveis pré-definidos.

Para incorporar os requisitos de QoC na aplicação móvel utilizamos a plataforma M-Hub/CDDL a qual contempla todos os parâmetros de QoC contidos no metamodelo. Um dos desafios para incorporar requisitos de QoC nessas aplicações é entender como funciona a plataforma de *middleware* utilizada, seus mecanismos para processamento de dados e como definir cada requisito necessário. Com isso, o mecanismo de transformação facilitou esse processo ao receber como entrada a especificação dos requisitos de QoC do sistema e gerar os artefatos de códigos necessários para garantir a QoC da aplicação e acompanhar o atendimento dos requisitos definidos. A geração automática do código reduz o tempo de desenvolvimento da aplicação, já que o desenvolvedor não precisará gerar os códigos relativos ao *middleware* M-Hub/CDDL manualmente. Logo o desenvolvedor teria que de forma manual realizar a codificação de cada parâmetro de QoC na plataforma de *middleware*.

O monitoramento dos requisitos foram realizados por meio de regras CEP geradas de forma automática pelo mecanismo de transformação. Com isso, as informações de contexto puderam ser avaliadas em tempo real para verificar se elas atendiam aos requisitos da aplicação. Para conceber essas regras o desenvolvedor teria que possuir conhecimentos necessários sobre o paradigma de programação CEP. Portanto, o mecanismo de transformação desenvolvido nesta pesquisa abstraiu essa complexidade, e gerou as regras CEP que foram utilizadas para avaliar se as informações de contexto disponibilizadas pelos provedores de contexto atendiam ou não aos requisitos de QoC especificados para aplicação.

O mecanismo de monitoramento forneceu um modelo de infraestrutura com um conjunto de ferramentas que facilitou o gerenciamento dos valores computados das métricas definidas para cada parâmetro de QoC especificado. A complexidade de criar as regras CEP que computaram os valores de cada métrica foi abstraída por meio do mecanismo de transformação.

Por meio dos *Dashboards* do Grafana, que permitiram a visualização das informações sobre as métricas, foi possível observar o estado atual e histórico das instâncias da aplicação que estavam em execução, e como elas se comportam ao longo do período de monitoramento em diversas situações que podem degradar a QoC da aplicação. Por exemplo, por meio do *Dashboard* em tempo real foi possível observar que a API de atividade física estava gerando muitas informações com um grau de confiança baixo conforme os requisitos da aplicação. Notamos também que o dispositivo vestível Polar H10, utilizado no estudo de

caso, estava gerando algumas informações de frequência cardíaca e ECG que não atendiam a taxa de atualização especificada.

O estudo de caso desenvolvido possui diversos requisitos e características típicas de aplicações de fenotipagem digital. Em resumo, essas aplicações focam na coleta de dados oriundas principalmente de sensores, distribuição dessas informações e inferência de situações. Por meio da solução proposta desenvolvida é possível, ao realizar a aquisição de informações de contexto, definir quais informações irão ser coletadas, de qual provedor de contexto e quais os requisitos necessários que devem ser atendidos para que a aplicação receba esse dado dos provedores de contexto. O mesmo acontece na distribuição das informações de contexto. Ao definir quais informações devem ser disseminadas, é possível especificar requisitos que garantem tanto a qualidade da informação que será enviada, como também a sua entrega para aplicações consumidoras. Desta forma, consideramos que os benefícios observados no desenvolvimento do estudo de caso realizado estarão também presentes no desenvolvimento de outras aplicações de fenotipagem digital.

6 Avaliação Experimental do uso de QoC no Estudo de Caso Desenvolvido

Neste capítulo demonstramos a eficiência ao utilizar a solução proposta por meio de uma avaliação experimental, onde foi possível analisar o impacto do mecanismo de QoC incorporado na aplicação desenvolvida como estudo de caso. Para esse propósito, foram realizados experimentos para verificar como a aplicação se comportaria com e sem o suporte a QoC em uma mesma situação para os seguintes requisitos, são eles: (i) identificar com exatidão atividades físicas que o usuário realizou; (ii) notificar aplicações clientes sobre a situação da conexão dos dispositivos vestíveis com o *smartphone*; (iii) garantir a entrega dos dados mesmo em situações de conexão intermitente; (iv) minimizar o consumo de bateria do *smartphone* e reduzir o volume de dados trafegados pela rede.

6.1 Avaliação dos Mecanismos de QoC

Para avaliação dos mecanismos de QoC a aplicação *mobile* Android foi instalada em um *smartphone* Redmi Note 8 com 4GB de memória RAM e sistema operacional Android 10. Os demais componentes de *software* Servidor, *Broker* MQTT, *Dashboard* e o sistema de monitoramento de QoC foram executadas em contêineres *Docker* em um notebook com processador Intel Core i7 1.8 Ghz com 20 GB de memória RAM e sistema operacional Ubuntu 22.04 LTS. O *smartphone* utilizado para executar a aplicação móvel estava conectado a uma rede Wi-Fi local. O dispositivo vestível Polar H10 foi utilizado para coleta dos dados.

Após a instalação da aplicação Android foram realizados dois experimentos. O primeiro deles avaliou com relação à identificação com exatidão das Atividades Físicas que o usuário realizou, notificação sobre a conexão dos dispositivos vestíveis com o *smartphone* e a entrega das informações de contexto mesmo em situações de conexão intermitente. Já o segundo experimento avaliou o consumo de bateria do *smartphone* e o volume de tráfego de informação de contexto enviadas por meio da infraestrutura de rede.

Para o primeiro experimento, foi realizada a coleta de dados de frequência cardíaca, ECG, atividade física, pressão arterial, SpO2 e glicose de um indivíduo por cerca de 8h para geração de um *dataset*. A Figura 21 apresenta uma amostra dos dados de frequência cardíaca extraída do *dataset* criado. Ele é formado por cinco atributos, são eles: (i) o *timestamp* representa o horário que a informação foi mensurada; (ii) o *sensorServiceName* indica o nome do dispositivo e o sensor; (iii) o *serviceName* significa o tipo de informação de contexto; (iv) o *avaliableAttributes* indica as propriedades dos valores que estão disponíveis

em cada medição; (v) o *serviceValue* indica o valor da medição.

	timestamp	sensorServiceName	serviceName	availableAttributes	serviceValue
2	2023-03-18T08:30:00.164Z	PolarH10_HR	heart rate	hr	75.0
3	2023-03-18T08:30:01.155Z	PolarH10_HR	heart rate	hr	74.0
4	2023-03-18T08:30:02.284Z	PolarH10_HR	heart rate	hr	74.0
5	2023-03-18T08:30:04.226Z	PolarH10_HR	heart rate	hr	74.0
6	2023-03-18T08:30:05.167Z	PolarH10_HR	heart rate	hr	74.0
7	2023-03-18T08:30:06.185Z	PolarH10_HR	heart rate	hr	75.0
8	2023-03-18T08:30:07.176Z	PolarH10_HR	heart rate	hr	76.0
9	2023-03-18T08:30:08.198Z	PolarH10_HR	heart rate	hr	75.0
10	2023-03-18T08:30:09.186Z	PolarH10_HR	heart rate	hr	75.0

Figura 21 – Amostras dos dados de Frequência Cardíaca

O *dataset* resultante foi utilizado como fonte de dados, onde foi criado na aplicação Android um serviço para ler o *dataset* e simular um monitoramento de saúde com base no envio dos dados já coletados para um ambiente com e sem suporte a QoC.

Já o segundo experimento objetivou avaliar o consumo de bateria do *smartphone* e o volume de tráfego na rede. O experimento foi dividido em duas etapas. A primeira delas foi realizar o teste sem o suporte a QoC. Os dados eram enviados para o servidor à medida que iam sendo gerados pelos sensores. Já para a segunda etapa do experimento, os dados foram agrupados por um período e enviados de uma só vez. Os períodos definidos para esse experimento foram de 15, 30 e 60 segundos.

Para cada momento do experimento foi realizado o monitoramento de um indivíduo por um período de 4 horas. O *smartphone* utilizado permaneceu o tempo todo com a tela desligada e apenas os processos necessários para execução do sistema operacional foram mantidos ativos. Após cada período, o *smartphone* era recarregado até completar os 100% da bateria. Para validar essa abordagem foram coletados dados referente ao consumo de bateria e o consumo em bytes dos dados que foram enviados.

6.1.1 Identificação das atividades físicas realizadas pelo usuário

Para avaliar a precisão das inferências das situações com relação às informações de atividade física, o usuário realizou anotações do início e fim de cada atividade realizada. Essas anotações foram necessárias para avaliar se a aplicação identificou corretamente a atividade que foi realizada naquele período informado, já que essas informações possuem um grau de confiança que varia de 0 a 100%. A Tabela 5 apresenta os períodos em que o usuário realizou determinadas atividades físicas e a duração de cada atividade.

Para analisar se as atividades físicas realizadas pelo usuário foram identificadas com exatidão verificou-se a similaridade entre as seções de atividade física identificadas, nos ambientes com e sem suporte a QoC, com as anotações descritas na Tabela 5. Para isso

Tabela 5 – Registro dos períodos de atividade física

Atividade	Horário inicial	Horário final	Duração (min)
Caminhando	13h:20	13h:30	10
Correndo	13h:30	13h:35	5
Parado	13h:35	13h:50	15
Caminhando	13h:50	13h:55	5
Parado	13h:55	14h:00	5
Caminhando	14h:00	14h:10	10
Correndo	14h:10	14h:15	5
Parado	14h:15	14h:20	5
Caminhando	14h:20	14h:30	10

utilizou-se o índice de Jaccard (Equação 6.1) que mensura a proporção entre a interseção dos intervalos encontrados e a sua união.

$$Jaccard(A, B) = \frac{A \cap B}{A \cup B} \tag{6.1}$$

Para identificar as seções de atividade física realizadas pelo usuário, implementamos o código apresentado na listagem Algoritmo 1. Este algoritmo recebe como dados de entrada a lista de atividades inferidas através da *Activity Recognition API* da Google ordenada de forma crescente por seus respectivos *timestamps*. Ele gera como saída uma lista de seções de atividades físicas realizadas contendo, para cada seção, o identificador da atividade, bem como o *timestamp* de início e fim da sua realização. Para identificar cada seção de atividade física, o algoritmo itera sobre a lista de atividades inferidas fornecida como entrada e, enquanto a atividade inferida for a mesma, ele mantém o registro de uma mesma seção. Ao encontrar uma atividade inferida diferente, uma nova seção de atividade física é aberta, contendo como *timestamp* de início o *timestamp* associado à nova atividade inferida. O *timestamp* que indica o fim de uma seção de atividade física corresponderá ao *timestamp* da última atividade física inferida da sequência com o mesmo indenfificador de atividade.

Na aplicação sem suporte a QoC não é foi possível filtrar as informações que possuem um grau de confiança adequado, definido como 70%, para identificar de maneira precisa as atividades realizadas pelo usuário. Com isso, podemos observar na Figura 22 que não foi possível identificar os períodos e as atividades que o usuário relatou na Tabela 5. Ao aplicar o índice de *Jaccard* encontrou-se um coeficiente de similaridade 0,26 que indica que não foi possível identificar com exatidão as seções de atividade com base nos dados que foram obtidos por meio da aplicação sem suporte a QoC.

Já as sessões de atividade física na aplicação com suporte a QoC foi definido no `Subscriber` que recebe as informações de atividade o parâmetro `Confidence`, onde as

Algoritmo 1: Algoritmo utilizado para identificar as seções de atividade física realizadas pelo usuário.

Data: $\{a_1, a_2, \ldots, a_n\}$ onde a_n é a enésima atividade identificada.
Result: $\{s_1, s_2, \ldots, s_n\}$ onde s_n é a enésima seção identificada.

```
1  atividadesIdentificadas ← {a_1, a_2, ..., a_n};
2  secoesIdentificadas ← {};
3  atividades ← {};
4  atividades ← atividades ∪ {atividadesIdentificadas[0]};
5  for i de 1 até tamanho(atividadesIdentificadas) do
6      atividadeAtual = atividadesIdentificadas[i] ;
7      if i <> tamanho(atividadesIdentificadas) - 1 then
8          indiceAtividadeFinal = tamanho(atividadesIdentificadas) − 1;
9          atividadeFinal = atividadesIdentificadas[indiceAtividadeFinal];
10         if atividadeAtual = atividadeFinal then
11             atividades ← atividades ∪ {atividadeAtual};
12         else
13             secao ← criarSecao(atividades) ;
14             secoesIdentificadas ← secoesIdentificadas ∪ {secao} ;
15             atividades ← {} ;
16             atividades ← atividades ∪ {atividadeAtual};
17         end
18     else
19         if atividades <> vazio then
20             atividades ← atividades ∪ {atividadeAtual};
21             secao ← criarSecao(atividades) ;
22             secoesIdentificadas ← secoesIdentificadas ∪ {secao} ;
23         end
24     end
25 end
```

informações que seriam recebidas pela aplicação e enviadas para o servidor seriam as que possuíam acima de 70% de confiança, assim garantimos que as informações utilizadas seriam informações precisas. Com isso, ao utilizar o algoritmo definido para identificar as sessões de atividade, com base nos dados coletados considerando o suporte a QoC, observamos na Figura 23 que foi possível identificar com exatidão as atividades realizadas pelo indivíduo ao confrontar as informações do gráfico com as anotações realizadas por ele (Tabela 5). Ao aplicar o índice de *Jaccard* encontrou-se um coeficiente de similaridade de 0,82 que representa uma correspondência significativa entre as atividades realizadas pelo usuário e as identificadas pelo algoritmo com base nos dados que possuem um nível de confiança igual ou maior que 70%.

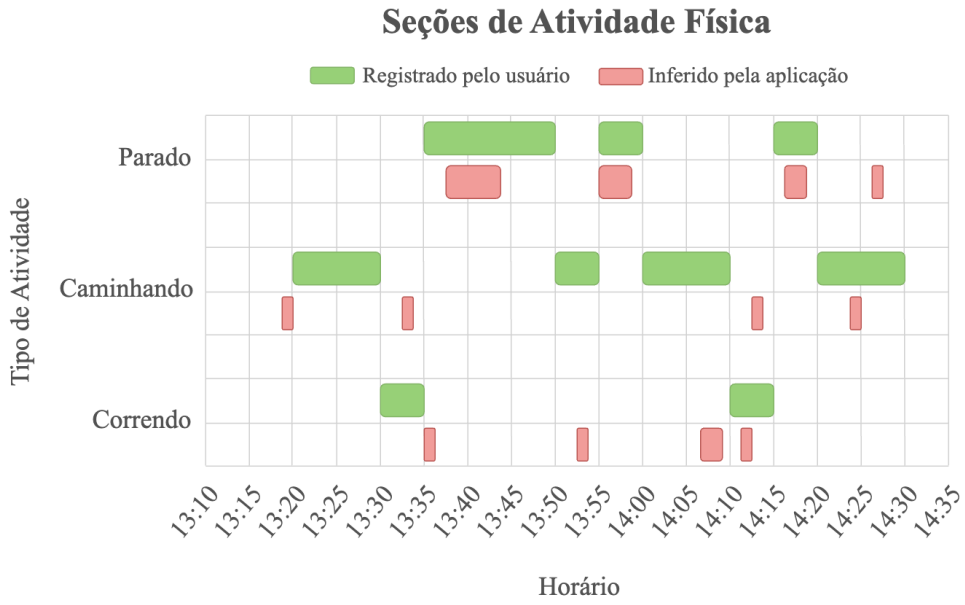


Figura 22 – Sessões identificadas com base nas informações de atividade física geradas pela *Activity Recognition API* e coletadas por meio da aplicação sem suporte a QoC.

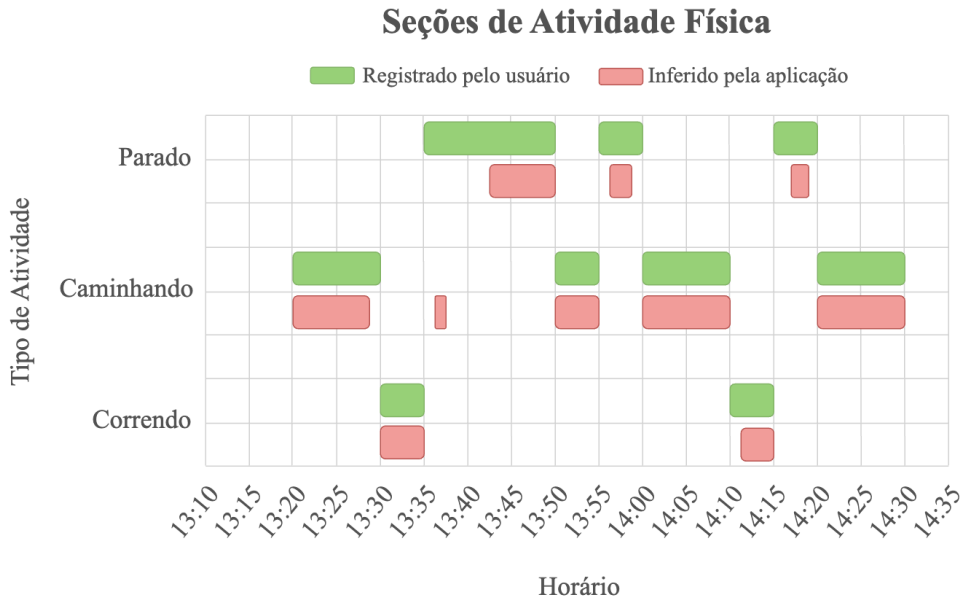


Figura 23 – Sessões identificadas com base nas informações de atividade física geradas pela *Activity Recognition API* e coletadas por meio da aplicação com suporte a QoC, na qual foi definido o parâmetro `Confidence`.

6.1.2 Desconexão dos dispositivos vestíveis com o *smartphone*

Para avaliar com relação à desconexão do dispositivo vestível com o *smartphone* foram simuladas desconexões entre o Polar H10 e o *smartphone*, onde o usuário monitorado afastava-se do *smartphone* em determinados horários. Para verificar se a aplicação identificou os períodos nos quais o *smartphone* manteve conexão ativa com o Polar H10, o usuário

registrou os horários nos quais o Polar H10 perdeu a conexão com o *smartphone*. A Tabela 6 apresenta o horário inicial, final e o tempo de duração do período da desconexão do *smartphone* com o dispositivo vestível Polar H10.

Tabela 6 – Períodos de desconexão do *smartphone* com o dispositivo vestível Polar H10

Horário inicial	Horário final	Duração (min)
09h:43	09h:46	3
10h:15	10h:25	10
13h:45	13h:50	5
14h:35	14h:42	7

Esse problema de desconexão do dispositivo vestível resulta em uma perda de dados que é inevitável. Mas para sabermos se essa perda de dados foi oriunda da desconexão do vestível com o *smartphone* a aplicação identifica os períodos de `conexão` e `desconexão` do dispositivo com o *smartphone*, pois foi definido para os `Publishers` que enviam dados oriundos do Polar H10 o parâmetro `Liveness`. A Figura 24 apresenta os períodos em que o dispositivo Polar H10 ficou conectado e desconectado com o *smartphone*. Ao aplicar o índice de *Jaccard* (Equação 6.1) encontrou-se um coeficiente de similaridade de 1 que representa uma correspondência exata entre os períodos de conexão e desconexão entre o Polar H10 e o *smartphone* anotados pelo usuário e os identificados pela aplicação.

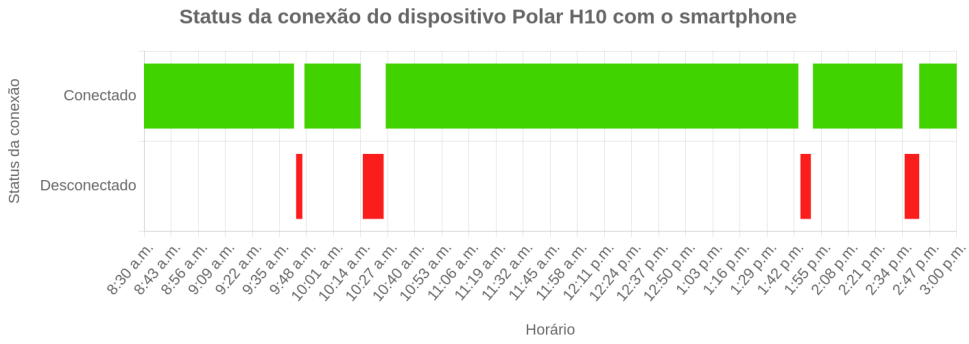


Figura 24 – Períodos de conexão e desconexão do Polar H10 com o *smartphone* identificados ao definir o parâmetro `Liveness`.

6.1.3 Garantia da entrega do dado

Sobre a perda de dados devido à perda de conexão com o servidor de *Broker*, foram simuladas desconexões no qual o usuário habilitava o modo avião do *smartphone*. Para verificar se a aplicação conseguiu garantir a entrega dos dados mesmo considerando uma conexão intermitente, o usuário registrou os períodos que ocorreram as desconexões. O tempo máximo configurado para o suporte a desconexão foi de 20 minutos. A Tabela 7 apresenta os períodos de desconexões simulados e a duração de cada período.

Tabela 7 – Período de desconexões do *smartphone* com o servidor de *Broker*

Horário inicial	Horário final	Duração (min)
09h:10	09h:25	15
09h:55	10h:00	5
10h:43	10h:46	3
10h:50	10h:53	3
12h:35	12h:45	10
13h:20	13h:30	10
14h:00	14h:10	10
14h:20	14h:25	5

Para a coleta manual dos dados, na qual se refere aos dados de pressão arterial, glicose e SpO2, o usuário realizou anotações dos horários em que os dados foram inseridos na aplicação. Essas anotações foram necessárias para realizar uma análise com relação ao recebimento de dados pelas aplicações consumidoras para verificar se a perca de pacotes da rede impactou no recebimento dos dados. A Tabela 8 apresenta os registros de horários e valores para os dados coletados de forma manual e inseridos na aplicação.

Tabela 8 – Registro de dados manuais

Horário	Tipo de dado	Medição
08h:30	Pressão arterial	sistólica: 120 mmHg, diastólica: 80 mmHg
08h:30	Glicose	95 mg/dL
08h:30	SpO2	99%
12h:40	Pressão arterial	sistólica: 130 mmHg, diastólica: 40 mmHg
12h:40	Glicose	98 mg/dL
12h:40	SpO2	100%
16h:30	Pressão arterial	sistólica: 130 mmHg, diastólica: 80 mmHg
16h:30	Glicose	110 mg/dL
16h:30	SpO2	99%

Para avaliar a com relação a perda de dados foi definida a métrica $Data_{loss}$ (Equação 6.2) que calcula a porcentagem de perda de dados com base no total de dados que foram enviados ($TData_{sent}$ e no total de dados recebidos $TData_{received}$.

$$Data_{loss} = 100 - (\frac{TData_{received} * 100}{TData_{sent}}) \tag{6.2}$$

A Tabela 9 apresenta a porcentagem da perda de dados ocasionadas pelas desconexões do *smartphone* com o servidor de *Broker* para o cenário onde a aplicação não possuía suporte a QoC. Os gráficos das Figuras 25 e 26 apresentam o impacto da perda de dados

com relação aos dados coletados dos sensores de frequência cardíaca e ECG. Por meio dos gráficos é possível visualizar os períodos nos quais tiveram perda de dados. Vale ressaltar que as imagens também apresentam os períodos nos quais a perda de dados foi devido a perda de conexão do dispositivo vestível Polar H10 com o *smartphone*.

Tabela 9 – Porcentagem de perda de dados no cenário sem suporte a QoC

Tipo de Dado	Porcentagem de perda
Frequência Cardíaca	17,18%
ECG	17,10%
Atividade Física	27,71%
Pressão Arterial	33,30%
Glicose	33,30%
Saturação	33,30%

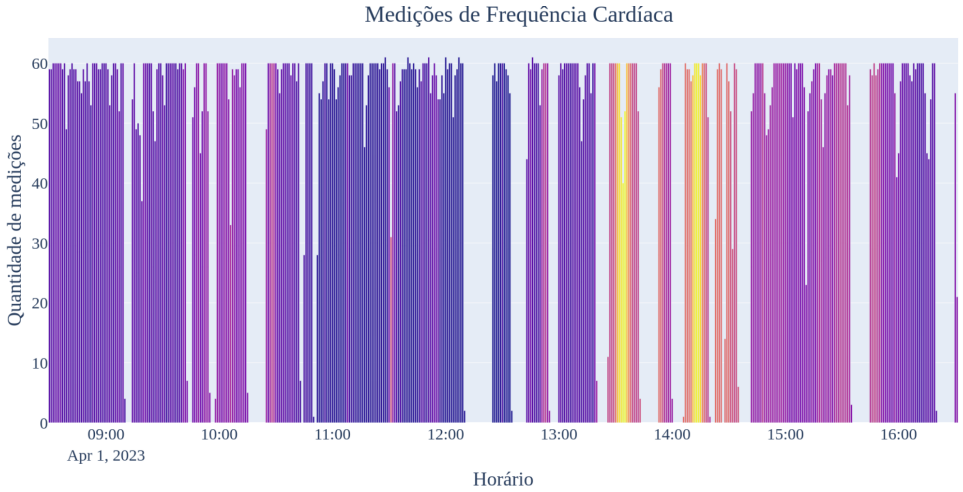


Figura 25 – Dados de frequência cardíaca coletados por meio da aplicação sem suporte a QoC

Os gráficos para as medições de frequência cardíaca e ecg apresentam a quantidade de medições que foram mensuradas em cada minuto. Nesse caso, considerando que os dados deveriam ser amostrados a cada 1 segundo, observamos que em alguns *slots* de 1 minuto não continha 60 medições. Investigamos o motivo pelo qual ocorriam estas perdas de dados: se decorrente da rede de computadores utilizada ou pela leitura dos dados nos sensores. Com isso realizamos um teste no qual os dados foram coletados diretamente dos sensores para verificar se a cada minuto eram geradas 60 leituras. Ao realizar esse teste notamos que nem sempre os sensores do Polar H10 geravam as leituras a cada segundo.

Mesmo com a perda de alguns dados de frequência cardíaca e ecg, ainda temos uma quantidade significativa de informações para realizar uma tomada de decisão. Já para os dados coletados de forma manual a perda de dados impacta de forma negativa, já

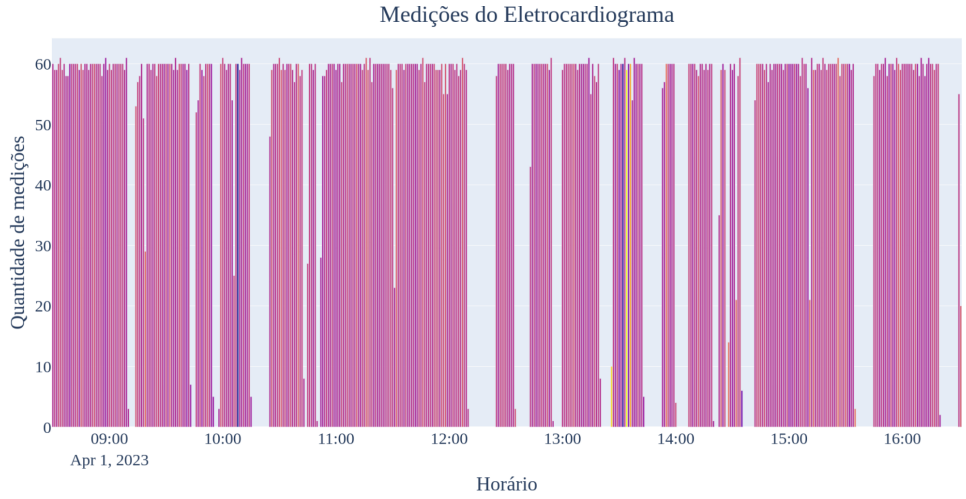


Figura 26 – Dados de ecg coletados por meio da aplicação sem suporte a QoC

que essas leituras são realizadas apenas algumas vezes ao dia. Sem a garantia de entrega dessa informação, podemos perder uma medição importante que apresenta o real estado de saúde do indivíduo naquele momento. Das três medições realizadas durante o período de monitoramento, só foram entregues para o servidor e persistida na base de dados duas medições dos dados de de pressão arterial, SpO2 e glicose.

Podemos observar que em um ambiente no qual não possui mecanismos que garantem a QoC da aplicação podemos ter vários problemas que impactam de forma negativa com relação à qualidade das informações coletadas para tomada de decisão. Já quando definimos na aplicação o parâmetro `History` notamos que mesmo em períodos de longas desconexões com o servidor a aplicação conseguiu garantir que os dados fossem entregues como mostra as Figuras 27 e 28.

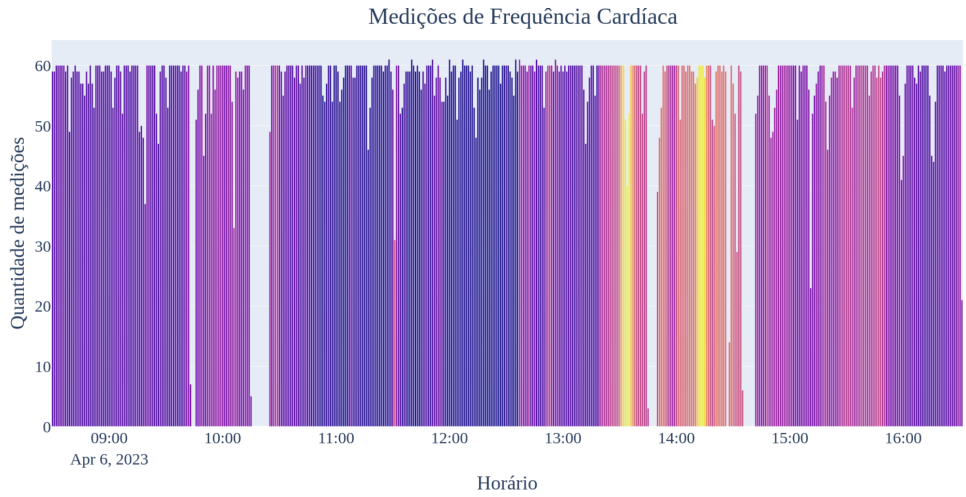


Figura 27 – Dados de frequência cardíaca coletados por meio da aplicação com suporte a QoC, na qual foi definido o parâmetro `History`.

Podemos observar também que nas Figuras 27 e 28 apresentam pequenos intervalos

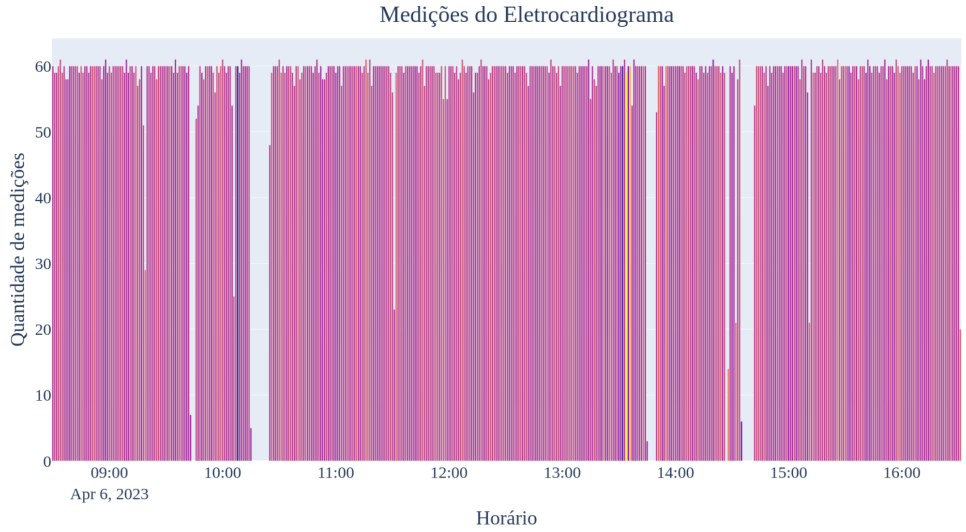


Figura 28 – Dados de ecg coletados por meio da aplicação com suporte a QoC, na qual foi definido o parâmetro `History`.

que representam perda de dados. Essa perda de dados é resultado de desconexões do *smartphone* com o dispositivo vestível Polar H10 que estava sendo utilizado pelo indivíduo monitorado, como podemos observar na Tabela 6. Logo, esse tipo de perda de dados é inavistável, já que o indivíduo pode se afastar do *smartphone* que está sendo utilizado para coletar dados do vestível. Para verificar se essa perda de dados foi oriunda da desconexão do vestível com o *smartphone* a aplicação identifica os períodos de `conexão` e `desconexão` entre o dispositivo vestível e o *smartphone*. A Tabela 10 apresenta a porcentagem de perda de dados ocasionada pela desconexão do dispositivo vestível Polar H10 com o *smartphone*.

Tabela 10 – Porcentagem de perda de dados no cenário com suporte a QoC, onde foi definido o parâmetro `History`.

Tipo de Dado	Porcentagem de perda
Frequência Cardíaca	7,20%
ECG	7,14%
Atividade Física	0%
Pressão Arterial	0%
Glicose	0%
Saturação	0%

6.1.4 Consumo de bateria do *smartphone* e volume de tráfego na rede

O experimento objetivou avaliar o consumo de bateria do *smartphone* e volume de tráfego na rede para um período de monitoramento, quando definimos o parâmetro `LatencyBudget`. Para isso, foram definidas algumas métricas. A métrica E_r calcula o consumo de bateria do *smartphone*. A equação referente a essa métrica é a seguinte:

$$E_r = E_i - E_f \tag{6.3}$$

onde E_i representa o nível de bateria do *smartphone* antes de iniciar o monitoramento, e E_f indica o nível de bateria ao final do monitoramento.

Já para verificar o volume de informações de contexto que foram trafegadas pela rede, a métrica N_c calcula a quantidade de *bytes* que foram trafegados pela rede. A equação referente a essa métrica é a seguinte:

$$N_c = \sum_{j=1}^{n} f_{bytes}(ci_j) \tag{6.4}$$

onde $f_{bytes}(ci_j)$ representa a quantidade de *bytes* de ci_j, sendo que sendo que $ci_j \in \theta = [ci_1, \ldots, ci_n]$ e ci_n é a enésima informação de contexto.

Ainda com relação ao volume de tráfego na rede, definimos a métrica G_{nc} para calcular a redução do volume de dados enviados pela rede ao definir o parâmetro `LatencyBudget`. A equação referente a essa métrica é a seguinte:

$$G_{nc} = 100 - (\frac{N_{c_{grouped}} * 100}{N_{c_{ungrouped}}}) \tag{6.5}$$

onde $N_{c_{grouped}}$ representa a quantidade de *bytes* trafegados na rede para a situação na qual os dados foram enviados de forma agrupada (ao definir o parâmetro `LatencyBudget`), e $N_{c_{ungrouped}}$ representa a quantidade de *bytes* que foram trafegados na rede quando os dados foram enviados a medida que iam sendo gerados (quando o parâmetro `LatencyBudget` não tinha sido definido).

Para o volume de tráfego na rede observamos que houve uma diferença significativa com relação à quantidade de dados enviados quando definimos o parâmetro `LatencyBudget`. Por exemplo, quando agrupados e comprimido os dados para um período de 60 segundos, tivemos uma redução de 91% em relação ao cenário no qual não definimos o parâmetro `LatencyBudget`. Isso reflete diretamente na redução do consumo de banda da rede utilizada para distribuição dos dados, pois o consumo de banda é proporcional a quantidade de bytes enviados pela rede. A Tabela 11 apresenta os resultados do volume de dados que foram trafegados pela rede.

Já com relação ao consumo de bateria do *smartphone* observamos uma redução do consumo em 1% a cada hora de monitoramento quando definimos uma Latência de 60 segundos. A Tabela 12 apresenta os resultados do consumo de energia.

Considerando o consumo de bateria quando definido a Latência de 60 segundos para um monitoramento de 4 horas em 16%, julgamos que para um período de monitoramento de 12h o consumo de bateria teria uma redução de 20% quando comparado ao cenário no qual

Tabela 11 – Volume de tráfego de rede

Etapa	Agrupamento (segundos)	Volume dados (MB)	Redução tráfego de dados
1ª	-	70,90	-
2ª	15	1,00	98,60%
3ª	30	0.50	99,30%
4ª	60	0,26	99,60%

Tabela 12 – Consumo de bateria

Etapas	Agrupamento (segundos)	Consumo bateria (%)
1ª	-	20
2ª	15	18
3ª	30	16
4ª	60	16

não se utiliza o parâmetro `LatencyBudget` para envio dos dados. Portanto, ao considerar o cenário no qual deseja-se monitorar o indivíduo por um longo período considerando as limitações de energia e de rede do *smatphone*, utilizar a abordagem de agrupamento e compressão para distribuição de dados trará um ganho significativo na economia desses recursos e permitirá que o processo de coleta e distribuição de dados ocorra por um maior período.

6.2 Considerações

A avaliação realizada através do estudo de caso implementado permitiu comprovar os benefícios da incorporação de requisitos. Foram realizados experimentos para verificar se a aplicação com suporte a QoC seria capaz de entregar as informações para as aplicações consumidoras mesmo que houvessem períodos de desconexão com o servidor, identificar períodos que os sensores dos dispositivos vestíveis ficaram ativos, distribuir apenas informações que possuem um grau de confiança aceitável para o contexto aplicação e reduzir o consumo de energia da bateria do *smartphone* e o volume de dados trafegados pela rede no processo de coleta e distribuição dos dados. Para isso, os testes foram realizados para duas instâncias da aplicação, uma que continha requisitos de QoC e a outra não.

A utilização de informações que não atendem a um certo nível de confiança é um problema para o processo de inferência de situações. Ao utilizar as informações de atividades físicas sem antes realizar uma seleção das informações que possuíam um grau de confiança ideal para o contexto da aplicação, não foi possível identificar as atividades físicas realizadas pelo usuário. Portanto, a utilização de filtros para uso apenas de informações de atividade física que atendam a um certo nível de confiança permitiu ao sistema inferir

com bom grau de acurácia a atividade realizada pelo usuário. Adicionalmente, este filtro reduz a quantidade de informações trafegadas pela rede e persistidas no banco de dados.

A perda de dados devido à não garantia da entrega pode resultar em dados faltantes de períodos que poderiam indicar alterações dos sinais vitais. Situações como essa podem impactar diretamente em ações que a aplicação deva realizar, como, por exemplo, a geração de alerta quando identificado alterações nos sinais vitais do indivíduo. Para o experimento no qual a aplicação não possuía mecanismos para garantir a QoC notamos que os dados distribuídos pela aplicação podem ser perdidos devido à perda de conexão com a internet ou com o servidor. Já na aplicação que possuía mecanismos para garantia da QoC, os dados foram persistidos no banco de dados local e posteriormente enviados quando a conexão foi restabelecida com o servidor.

Referente ao volume de tráfego de rede, quando realizamos o experimento sem especificar um controle de latência notamos que muitos dados foram enviados pela rede, aumentando o consumo de banda. Isso pode ser um problema quando o monitoramento é realizado por um longo período e, além disso, o usuário monitorado estiver utilizando uma rede móvel. Ao definir o controle de latência observamos uma redução significativa no consumo ao agrupar os dados e comprimi-los antes de trafegar essas informações pela rede. Notamos também uma redução no consumo de bateria do *smartphone* quando definimos o controle de latência.

A utilização de mecanismos de QoC se mostrou eficiente para garantir um nível de qualidade aceitável da aplicação. Com a incorporação do suporte a QoC na aplicação por meio do processo proposto nesta dissertação foi possível garantir: a exatidão na inferência de situações, pois os dados foram providos por uma fonte que possui um certo grau de imprecisão; a entrega das informações mesmo considerando conexão intermitente, ocasionadas pela mobilidade do usuário e pela tecnologia de rede utilizada; a redução do consumo de recursos, como bateria e de rede.

7 Conclusão

Este estudo propôs um processo para incorporar requisitos de QoC em aplicações de fenotipagem digital. O processo possui cinco etapas, a saber: (i) realizar a especificação dos requisitos de QoC por meio de um metamodelo; (ii) a geração de código alvo por meio de um mecanismo de transformação que recebe como entrada a especificação dos requisitos; (iii) a incorporação na aplicação dos códigos fonte gerados para garantir os requisitos de QoC; (iv) monitoramentos dos parâmetros de QoC; (iv) visualização do resultado do monitoramento por meio de *Dashboards* que visa fornecer ao desenvolvedor um ambiente no qual seja possível acompanhar o nível de qualidade das instâncias da aplicação.

Para especificação dos requisitos de QoC foi concebido um metamodelo considerando a estrutura de aplicações de fenotipagem digital na aquisição e distribuição de informações de contexto. Para etapa de transformação foi implementado um mecanismo que gera códigos na linguagem Java incorporados pelo desenvolvedor na aplicação desenvolvida com a plataforma de *middleware* M-Hub/CDDL. A etapa de transformação gera também classes, na linguagem Java, responsáveis por fornecer métodos para realizar a avaliação das informações de contexto e o monitoramento dos parâmetros de QoC especificados. O monitoramento é realizado por meio de métricas, definidas para cada parâmetro de QoC contidos no metamodelo, para mensurar o nível de qualidade da aplicação. Além dos códigos alvos gerados para serem incorporados na aplicação Android, o mecanismo de transformação gera os *Dashboards* utilizados para visualização dos resultados das métricas. Os *Dashboards* são especificados no formato JSON, e importados para a ferramenta de visualização Grafana.

Para avaliação do processo proposto foi desenvolvido como estudo de caso um *software* intitulado LSDi mHealth que monitora os sinais vitais e a atividade física dos usuários durante sua rotina por meio de dispositivos vestíveis. Para garantir a QoC da aplicação, o processo desenvolvido foi aplicado. O *software* possui três principais componentes. O primeiro é um aplicativo móvel que coleta informações de sinal e atividade. O segundo é um servidor que recebe as informações de contexto, realiza inferência de situações de interesse e armazena os dados em banco. Por fim, o terceiro é um *Dashboard* que fornece aos profissionais de saúde informações quase em tempo real e dados históricos sobre os sinais vitais dos usuários monitorados.

Por meio do estudo de caso desenvolvido, foi possível observar que o processo proposto foi capaz de auxiliar o desenvolvedor a expressar requisitos de QoC para aplicações de fenotipagem digital, a partir de metamodelo que possui um conjunto de parâmetros de QoC pré-definidos. O processo ajudou também na incorporação dos requisitos de QoC na

aplicação por meio de um mecanismo de geração de código-fonte, e no monitoramento em tempo real do cumprimento dos requisitos especificados.

A partir do desenvolvimento do estudo de caso foram realizados experimentos para verificar como a aplicação se comportaria com e sem o suporte a QoC em uma mesma situação. Após os experimentos realizados foi possível observar que a incorporação de requisitos de QoC em aplicações de fenotipagem digital no âmbito da saúde é benéfico com relação à garantia da qualidade das informações geradas por meio dessas aplicações.

7.1 Contribuições

As principais contribuições científicas desta pesquisa são:

- Um processo que norteia a incorporação de requisitos de QoC em aplicações de fenotipagem digital.
- Um metamodelo para especificar requisitos de QoC considerando a estrutura das aplicações de fenotipagem digital na aquisição e distribuição de informações de contexto.
- Um mecanismo de transformação que realiza a geração de código alvo para serem incorporados nas aplicações de fenotipagem digital desenvolvidas com base na plataforma de *middleware* M-Hub/CDDL.
- Definição de um conjunto de métricas para avaliar o nível de QoC da aplicação.
- Concepção de um ambiente para monitoramento dos requisitos de QoC em tempo real que também permite a consulta a dados históricos.

7.2 Trabalhos Futuros

A partir do trabalho desenvolvido, outras iniciativas de pesquisa e desenvolvimento podem ser conduzidas, como:

- Fornecer uma notação gráfica do metamodelo proposto para facilitar no processo de especificação dos requisitos de QoC.
- Permitir que as métricas sejam definidas na etapa de especificação dos requisitos da aplicação, possibilitando aos desenvolvedores escolherem apenas as métricas necessárias para o contexto da sua aplicação.
- Realizar uma avaliação de usabilidade do processo de incorporação de requisitos de QoC com desenvolvedores de softwares que atuam na concepção de aplicações de fenotipagem digital.

7.3 Publicações

Para divulgação dos resultados desta pesquisa, foi publicado o seguinte artigo:

- Luís Eduardo Costa Laurindo, Ivan Rodrigues de Moura, Luciano Reis Coutinho e Francisco José da Silva e Silva. *Specification of Quality of Context Requirements for Digital Phenotyping Applications*. In: 16th EAI *International Conference on Pervasive Computing Technologies for Healthcare* - EAI PervasiveHealth 2022, Thessaloniki, Greece.
  Tipo de publicação: Evento científico internacional.
  Qualis CAPES(2017-2020): A3
  Situação: O artigo foi publicado como capítulo de livro da série *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* (LNICST).
  DOI: <https://doi.org/10.1007/978-3-031-34586-9_43>

Referências

AHARONY, N.; PAN, W.; IP, C.; KHAYAL, I.; PENTLAND, A. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, Elsevier, v. 7, n. 6, p. 643–659, 2011. Citado 5 vezes nas páginas 18, 29, 36, 37 e 38.

ARNDT, B. F. Mme-mdd: um método para manutenção e evolução de sistemas baseados no mdd. Universidade Federal de São Carlos, 2016. Citado 3 vezes nas páginas 10, 27 e 28.

BELLAVISTA, P.; CORRADI, A.; FANELLI, M.; FOSCHINI, L. A survey of context data distribution for mobile ubiquitous systems. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 44, n. 4, p. 1–45, 2012. Citado na página 18.

BEZERRA, E. D. C.; TELES, A. S.; COUTINHO, L. R.; SILVA, F. J. da Silva e. Dempster–shafer theory for modeling and treating uncertainty in iot applications based on complex event processing. *Sensors*, MDPI, v. 21, n. 5, p. 1863, 2021. Citado 2 vezes nas páginas 10 e 24.

BOLCHINI, C.; CURINO, C. A.; ORSI, G.; QUINTARELLI, E.; ROSSATO, R.; SCHREIBER, F. A.; TANCA, L. And what can context do for data? *Communications of the ACM*, ACM New York, NY, USA, v. 52, n. 11, p. 136–140, 2009. Citado na página 21.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, Morgan & Claypool Publishers, v. 3, n. 1, p. 1–207, 2017. Citado na página 26.

BUCHHOLZ, T.; KÜPPER, A.; SCHIFFERS, M. Quality of context: What it is and why we need it. In: *Workshop of the HP OpenView University Association*. [S.l.: s.n.], 2003. Citado 2 vezes nas páginas 18 e 21.

CHO, S.; ENSARI, I.; WENG, C.; KAHN, M. G.; NATARAJAN, K. Factors affecting the quality of person-generated wearable device data and associated challenges: rapid systematic review. *JMIR mHealth and uHealth*, JMIR Publications Inc., Toronto, Canada, v. 9, n. 3, p. e20738, 2021. Citado 2 vezes nas páginas 18 e 22.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 44, n. 3, p. 1–62, 2012. Citado na página 23.

DÜKING, P.; FUSS, F. K.; HOLMBERG, H.-C.; SPERLICH, B. et al. Recommendations for assessment of the reliability, sensitivity, and validity of data provided by wearable sensors designed for monitoring physical activity. *JMIR mHealth and uHealth*, JMIR Publications Inc., Toronto, Canada, v. 6, n. 4, p. e9341, 2018. Citado na página 22.

EsperTech. *EsperTech: Complex Event Processing, Streaming Analytics*. 2022. Disponível em: <https://www.espertech.com/>. Acesso em: 13 abril 2022. Citado na página 24.

FERREIRA, D.; KOSTAKOS, V.; DEY, A. K. Aware: mobile context instrumentation framework. *Frontiers in ICT*, Frontiers, p. 6, 2015. Citado 6 vezes nas páginas 10, 18, 30, 36, 37 e 38.

GOMES, B. d. T. P.; MUNIZ, L. C. M.; SILVA, F. J. Da Silva e; SANTOS, D. V. D.; LOPES, R. F.; COUTINHO, L. R.; CARVALHO, F. O.; ENDLER, M. A middleware with comprehensive quality of context support for the internet of things applications. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 12, p. 2853, 2017. Citado 4 vezes nas páginas 10, 18, 21 e 25.

HARDY, J.; VEINOT, T. C.; YAN, X.; BERROCAL, V. J.; CLARKE, P.; GOODSPEED, R.; GOMEZ-LOPEZ, I. N.; ROMERO, D.; VYDISWARAN, V. V. User acceptance of location-tracking technologies in health research: implications for study design and data quality. *Journal of biomedical informatics*, Elsevier, v. 79, p. 7–19, 2018. Citado na página 23.

HAREL, D.; RUMPE, B. Modeling languages: Syntax, semantics and all that stu. *N/A n/a*, p. 1–28, 2000. Citado na página 26.

HICKS, J. L.; ALTHOFF, T.; SOSIC, R.; KUHAR, P.; BOSTJANCIC, B.; KING, A. C.; LESKOVEC, J.; DELP, S. L. Best practices for analyzing large-scale health data from wearables and smartphone apps. *NPJ digital medicine*, Nature Publishing Group, v. 2, n. 1, p. 1–12, 2019. Citado na página 22.

HUZOOREE, G.; KHEDO, K. K.; JOONAS, N. Data reliability and quality in body area networks for diabetes monitoring. In: *Body Area Network Challenges and Solutions*. [S.l.]: Springer, 2019. p. 55–86. Citado na página 23.

JAGARLAMUDI, K. S.; ZASLAVSKY, A.; LOKE, S. W.; HASSANI, A.; MEDVEDEV, A. Requirements, limitations and recommendations for enabling end-to-end quality of context-awareness in iot middleware. *Sensors*, MDPI, v. 22, n. 4, p. 1632, 2022. Citado 2 vezes nas páginas 18 e 21.

KARKOUCH, A.; MOUSANNIF, H.; MOATASSIME, H. A.; NOEL, T. Data quality in internet of things: A state-of-the-art survey. *Journal of Network and Computer Applications*, Elsevier, v. 73, p. 57–81, 2016. Citado na página 22.

KLEPPE, A. G.; WARMER, J. B.; BAST, W. *MDA explained: the model driven architecture: practice and promise*. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 27.

KOCH, N. Transformation techniques in the model-driven development process of uwe. In: *Workshop proceedings of the sixth international conference on Web engineering*. [S.l.: s.n.], 2006. p. 3–es. Citado na página 27.

MANZOOR, A.; TRUONG, H.-L.; DUSTDAR, S. Quality of context: models and applications for context-aware systems in pervasive environments. *The Knowledge Engineering Review*, Cambridge University Press, v. 29, n. 2, p. 154–170, 2014. Citado na página 21.

MELLOR, S. J.; SCOTT, K.; UHL, A.; WEISE, D.; DISTILLED, M. Principles of model-driven architecture. *Boston et al*, 2004. Citado na página 26.

MENDES, J. P.; MOURA, I. R.; VEN, P. Van de; VIANA, D.; SILVA, F. J.; COUTINHO, L. R.; TEIXEIRA, S.; RODRIGUES, J. J.; TELES, A. S. Sensing apps and public data sets for digital phenotyping of mental health: Systematic review. *Journal of medical*

Internet research, JMIR Publications Inc., Toronto, Canada, v. 24, n. 2, p. e28735, 2022. Citado 4 vezes nas páginas 17, 18, 29 e 37.

MENDES, J. P. M. et al. Um framework para facilitar o desenvolvimento de aplicações móveis de fenotipagem digital. Universidade Federal do Maranhão, 2022. Citado 6 vezes nas páginas 10, 18, 34, 35, 36 e 37.

MISHRA, B.; KERTESZ, A. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, IEEE, v. 8, p. 201071–201086, 2020. Citado na página 25.

MOURA, I.; TELES, A.; SILVA, F.; VIANA, D.; COUTINHO, L.; BARROS, F.; ENDLER, M. Mental health ubiquitous monitoring supported by social situation awareness: A systematic review. *Journal of Biomedical Informatics*, Elsevier, v. 107, p. 103454, 2020. Citado 2 vezes nas páginas 17 e 18.

OMETOV, A.; SHUBINA, V.; KLUS, L.; SKIBIńSKA, J.; SAAFI, S.; PASCACIO, P.; FLUERATORU, L.; GAIBOR, D. Q.; CHUKHNO, N.; CHUKHNO, O.; ALI, A.; CHANNA, A.; SVERTOKA, E.; QAIM, W. B.; CASANOVA-MARQUéS, R.; HOLCER, S.; TORRES-SOSPEDRA, J.; CASTELEYN, S.; RUGGERI, G.; ARANITI, G.; BURGET, R.; HOSEK, J.; LOHAN, E. S. A survey on wearable technology: History, state-of-the-art and current challenges. *Computer Networks*, v. 193, p. 108074, 2021. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128621001651>. Citado na página 17.

PARREIRAS, F. S. *Semantic Web and model-driven engineering*. [S.l.]: John Wiley & Sons, 2012. Citado na página 27.

RAHMANI, A. M.; BABAEI, Z.; SOURI, A. Event-driven iot architecture for data analysis of reliable healthcare application using complex event processing. *Cluster Computing*, Springer, v. 24, n. 2, p. 1347–1360, 2021. Citado na página 23.

REINERMAN-JONES, L.; HARRIS, J.; WATSON, A. Considerations for using fitness trackers in psychophysiology research. In: SPRINGER. *International Conference on Human Interface and the Management of Information*. [S.l.], 2017. p. 598–606. Citado na página 23.

SACCARO, L. F.; AMATORI, G.; CAPPELLI, A.; MAZZIOTTI, R.; DELL'OSSO, L.; RUTIGLIANO, G. Portable technologies for digital phenotyping of bipolar disorder: A systematic review. *Journal of Affective Disorders*, Elsevier, v. 295, p. 323–338, 2021. Citado na página 17.

SCHUELLER, S. M.; BEGALE, M.; PENEDO, F. J.; MOHR, D. C. Purple: a modular system for developing and deploying behavioral intervention technologies. *Journal of medical Internet research*, JMIR Publications Inc., Toronto, Canada, v. 16, n. 7, p. e3376, 2014. Citado 3 vezes nas páginas 18, 29 e 36.

SILVA, A. R. D. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, Elsevier, v. 43, p. 139–155, 2015. Citado na página 27.

SILVA, M.; TELES, A.; LOPES, R.; SILVA, F.; VIANA, D.; COUTINHO, L.; GUPTA, N.; ENDLER, M. Neighborhood-aware mobile hub: An edge gateway with leader election

mechanism for internet of mobile things. *Mobile Networks and Applications*, Springer, p. 1–14, 2020. Citado na página 24.

Statista. *Number of smartphone subscriptions worldwide from 2016 to 2027*. 2022. Disponível em: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>. Acesso em: 13 abril 2022. Citado na página 17.

TELES, A. S.; ROCHA, A.; SILVA, F. José da Silva e; LOPES, J. C.; O'SULLIVAN, D.; VEN, P. Van de; ENDLER, M. Enriching mental health mobile assessment and intervention with situation awareness. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 1, p. 127, 2017. Citado 5 vezes nas páginas 10, 32, 33, 36 e 37.

TOROUS, J.; KIANG, M. V.; LORME, J.; ONNELA, J.-P. et al. New tools for new research in psychiatry: a scalable and customizable platform to empower data driven smartphone research. *JMIR mental health*, JMIR Publications Inc., Toronto, Canada, v. 3, n. 2, p. e5165, 2016. Citado 7 vezes nas páginas 10, 17, 18, 32, 36, 37 e 38.

WISNIEWSKI, H.; HENSON, P.; TOROUS, J. Using a smartphone app to identify clinically relevant behavior trends via symptom report, cognition scores, and exercise levels: a case series. *Frontiers in psychiatry*, Frontiers Media SA, v. 10, p. 652, 2019. Citado 5 vezes nas páginas 10, 33, 34, 36 e 37.

XIONG, H.; HUANG, Y.; BARNES, L. E.; GERBER, M. S. Sensus: a cross-platform, general-purpose system for mobile crowdsensing in human-subject studies. In: *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing*. [S.l.: s.n.], 2016. p. 415–426. Citado 5 vezes nas páginas 10, 18, 31, 36 e 37.