UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Luiz Carlos Melo Muniz

Avaliação e Monitoramento de QoC em Sistemas Cientes de Contexto

São Luís (MA)
2017

Luiz Carlos Melo Muniz

Avaliação e Monitoramento de QoC em Sistemas Cientes de Contexto

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Ciência da Computação com área de concentração em Ciência da Computação.

Orientador: Francisco José da Silva e Silva

Doutor - UFMA

São Luís (MA)

2017

Muniz, Luiz Carlos Melo

Avaliação e Monitoramento de QoC em Sistemas Cientes de Contexto / Luiz Carlos Melo Muniz. – São Luís (MA), 2017.

112 f.

Orientador: Francisco José da Silva e Silva.

Impresso por computador (fotocópia).

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Ciência da Computação. São Luís (MA), 2017.

1. Internet das Coisas. 2. Qualidade de Contexto. 3. Middleware Cientes de Contexto. 4. Avaliação de QoC. 5. Monitoreamento de QoC I. Silva, Francisco José da Silva e, orient. II. Título.

CDU

Dedico esta conquista aos meus pais que com muito amor me educaram e incentivaram em todos os momentos da minha vida.

Resumo

Qualidade de Contexto (QoC) é um conceito que pode ser utilizado para caracterizar tanto a qualidade da informação (QoI) quanto a qualidade do serviço de distribuição dos dados de contexto (QoS). Para que sejam capazes de atender também requisitos de QoC, os sistemas de *middleware* de contexto convencionais precisam ser estendidos. Isso significa que suas interfaces, serviços e arquiteturas devem ser modificados para considerar novos requisitos de qualidade de contexto. Algumas aplicações da IoT podem apresentar requisitos de QoC mais complexos que outras. Essa complexidade é maior quando as aplicações requerem múltiplos parâmetros de QoC considerando ambas dimensões (QoI e QoS), que nem sempre são suportadas pela camada de *middleware*. Adicionalmente, um suporte mais abrangente da QoC requer por parte do *middleware* funcionalidades como: avaliação de parâmetros de QoC, consultas, descoberta de serviços, filtragem e o monitoramento de informações de contexto com base em QoC. Uma solução de *middleware* capaz de lidar com todos esses desafios ainda não está disponível. Com o objetivo de contribuir com o estado da arte no desenvolvimento de *middleware* com suporte a QoC, este trabalho de mestrado propõe componentes capazes de avaliar, filtrar e monitorar a QoC com suporte a diversos parâmetros de qualidade dos dados/sensores.

Palavras-chaves: Internet das Coisas, Qualidade de Contexto, Middleware Cientes de Contexto, Avaliação de QoC, Monitoramento de QoC.

Abstract

Quality of Context (QoC) is a concept that can be used to characterize both the quality of information (QoI) and the quality of the context data distribution service (QoS). In order to be able to also meet QoC requirements, conventional context middleware systems need to be extended. This means that their interfaces, services, and architectures must be modified to consider new context-quality requirements. Some IoT applications may have more complex QoC requirements than others. This complexity is greater when applications require multiple QoC parameters considering both dimensions (QoI and QoS), which are not always supported by the middleware layer. In addition, more comprehensive QoC support requires middleware functionality such as QoC parameter evaluation, querying, service discovery, filtering, and QoC-based context information monitoring. A middleware solution able to handle all these challenges is not yet available. With the objective of contributing to the state of the art in the development of middleware with QoC support, this master's work proposes components capable of evaluating, filtering and monitoring the QoC with support to several parameters of data quality/sensors.

Keywords: Internet of Things, Quality of Context, Context-Aware Middleware, QoC Evaluation, QoC Monitoring.

Agradecimentos

Em primeiro lugar, a Deus, pelo dom da vida. Ele conhece bem as minhas limitações e me permitiu superá-las em diversos momentos da minha trajetória até aqui.

Ao meu orientador, Prof. Francisco Silva, pela confiança, disponibilidade, e paciência.

Ao coordenador, colegiado, professores e técnicos administrativos do Programa de Pós-Graduação em Ciência da Computação da UFMA, pelo trabalho sério e dedicado, que possibilita as condições para o desenvolvimento de nossas pesquisas.

Aos membros da banca examinadora, por aceitarem a missão de avaliar este trabalho.

Aos meus colegas do Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da UFMA, por compartilharem comigo momentos bons e ruins nesta pós-graduação.

Aos pesquisadores do *Laboratory for Advanced Collaboration* da Pontifícia Universidade Católica do Rio de Janeiro (PUC), parceiros com os quais pude buscar ajuda e co-orientação, trocar experiências valiosas, desenvolver projetos e publicar artigos.

A minha família pelo apoio em todos os momentos.

"O segredo do sucesso é a constância do propósito."

Benjamin Disraeli

Sumário

Lista de Figuras **xii**

Lista de Tabelas **xiv**

1 Introdução **14**

1.1 Contextualização 14

1.2 Motivação e Caracterização do Problema 16

1.2.1 Objetivos 19

1.3 Organização do Trabalho 20

2 Fundamentação Teórica **21**

2.1 Ciência de Contexto 21

2.1.1 Classificação de Contexto 22

2.1.2 Modelos de Representação de Contexto 23

2.1.3 Ciclo de Gerenciamento de Contexto 26

2.1.4 Modelos de Distribuição de Informações de Contexto 27

2.2 Qualidade de Contexto 28

2.2.1 Definições de QoC 29

2.2.2 Parâmetros de QoC 30

2.2.3 Parâmetros de Qualidade das Informações/Sensores 30

2.2.4 Parâmetros de Qualidade do Serviço de Distribuição 35

2.2.5 Motivação para o uso da QoC 37

2.2.6 Fatores que degradam a Qualidade de Contexto 38

2.3 Monitoramento da QoC 39

2.4 Processamento de Eventos Complexos 41
2.4.1 Agentes de Processamento de Eventos e Redes de Processamento de Eventos 42
2.4.2 Linguagem de Processamento de Eventos 43
2.5 M-Hub 45

3 Solução Proposta **50**

3.1 Requisitos do CDDL 51
3.2 Arquitetura e Funcionalidades dos Componentes 53
3.3 Interfaces e Abstrações de Programação 56
3.3.1 Consulta aos Serviços Disponíveis por Smart Objects 57
3.3.2 Consumindo Dados de Contexto 58
3.3.3 Publicando Dados de contexto 62
3.3.4 Uso de filtros 64
3.3.5 Monitorando serviços e dados de contexto 65
3.4 Aspectos de Implementação dos Componentes QoCEvaluator, Monitor e Filter 67
3.4.1 QoC Evaluator 67
3.4.2 Filter 71
3.4.3 Monitor 73

4 Avaliação dos Mecanismos Propostos **76**

4.1 Avaliação de Desempenho do Mecanismo de Monitoramento 76
4.2 Avaliação do Consumo de Memória dos Mecanismos 78
4.2.1 Avaliação do Consumo de Bateria 79

5 Trabalhos Relacionados e Análise Comparativa **80**

5.1 QoMonitor 80
5.2 AWARENESS 83

5.3 COSMOS 86
5.4 COPAL 89
5.5 INCOME 92
5.6 SALES 93
5.7 Análise dos Trabalhos Relacionados 97
5.7.1 Suporte à Avaliação de QoC 97
5.7.2 Quantidade de Parâmetros Suportados 98
5.7.3 Suporte ao Monitoramento de QoC 99
5.7.4 Linguagem de especificação de Requisitos de Monitoramento 100

6 Conclusões **101**

6.1 Trabalhos Futuros 104

Referências Bibliográficas **105**

Lista de Figuras

2.1 Classificação de Contexto segundo Emmanouilidis et al. 23

2.2 Rede de Processamento de Eventos 44

2.3 Arquitetura do M-Hub 48

3.1 Arquitetura estendida do M-Hub/CDDL 54

3.2 Interface Subscriber 59

3.3 Interface Publisher 63

3.4 Exemplo de filtragem 64

3.5 Interface Filter 65

3.6 Interface Monitor 66

3.7 Diagrama de classes do QoC Evaluator 70

3.8 Diagrama de sequência do QoC Evaluator 71

3.9 Diagrama de classes do Filter 72

3.10 Diagrama de sequência do processo de filtragem 73

3.11 Diagrama de classes do Monitor 74

3.12 Diagrama de sequência do processo de monitoramento 75

4.1 Fluxo de dados da aplicação teste 77

5.1 Arquitetura Geral do QoMonitor 81

5.2 Arquitetura Geral do AWARENESS [79] 84

5.3 Componentes do CMS do AWARENESS [79] 86

5.4 Arquitetura do *Context Node* do COSMOS [1] 87

5.5 Arquitetura do *QoC Context Node* do COSMOS [1] 88

5.6 Arquitetura do *QoC Operator* do COSMOS [1] 88
5.7 Arquitetura do COPAL [50] 90
5.8 Arquitetura do INCOME [61] 92
5.9 Arquitetura Lógica do SALES [22] 94
5.10 Arquitetura de Software do SALES [22] 95

Lista de Tabelas

4.1 Tempo que a aplicação leva para detectar uma mudança na QoC 78

4.2 Tempo de processamento da mensagem pelo *Monitor* 78

4.3 Consumo de Memória em Kbytes 79

4.4 Consumo de Bateria 79

5.1 Comparativo entre os trabalhos relacionados e o M-Hub/CDDL 97

1 Introdução

1.1 Contextualização

A computação ciente de contexto refere-se à capacidade de um sistema computacional perceber características do meio ambiente que sejam de seu interesse e tomar decisões de acordo com mudanças nesse ambiente. Aplicações sensíveis ao contexto são aquelas capazes de coletar, extrair e utilizar as informações de contexto com a finalidade de adaptar o seu comportamento e executar ações em resposta às mudanças do ambiente, sem que intervenções sucessivas dos usuários sejam necessárias [73]. Tais aplicações reagem a ações executadas por outras entidades, podendo estas serem pessoas, objetos ou até mesmo outros sistemas, que modifiquem o ambiente. A computação ciente de contexto visa tornar a experiência do usuário em relação à utilização de recursos computacionais mais agradável. Em cenários de computação ubíqua, o foco dos usuários seria a tarefa e não a ferramenta utilizada, e eles nem sequer perceberiam ou necessitariam de conhecimentos técnicos relativos aos recursos computacionais utilizados [21]. Para que este nível de invisibilidade seja atingido, é fundamental que as aplicações sejam capazes de obter informações do ambiente no qual o usuário se encontra a fim de fornecer a ele serviços e informações úteis.

Nos últimos anos, um novo paradigma conhecido como Internet das Coisas (IoT – *Internet of Things*) tem sido associado à computação ciente de contexto como forma de se obter uma melhor percepção dos contextos ambientais (físicos) nos quais as aplicações executam. A IoT é um campo de pesquisa que integra aspectos e tecnologias de diferentes áreas como computação ubíqua, ciência de contexto, protocolos e tecnologias de comunicação, redes e dispositivos com sensores embutidos. Essa integração visa gerar um sistema dinâmico global onde milhares de dispositivos endereçáveis e heterogêneos, conhecidos como objetos inteligentes (*smart objects*), possuem uma representação na Internet e são capazes de compartilhar dados de contexto entre si e com aplicações diversas [9]. Esses objetos podem ter recursos

de processamento e conectividade limitados que precisam de um servidor local com acesso à Internet, conhecido como *gateway* da IoT, para que possam enviar dados para as aplicações remotas. Uma extensão da IoT chamada de Internet das Coisas Móveis (IoMT - *Internet of Mobile Things*) propõe cenários onde objetos podem ser movidos ou mover-se de forma autônoma, mas ainda assim permanecerem acessíveis remotamente a partir de um *gateway* também móvel. Exemplos de objetos móveis incluem dispositivos vestíveis ou portáteis, robôs e veículos com sensores embutidos [53].

A computação ciente de contexto e a IoT têm sido exploradas nos mais diversos domínios de aplicação, como os setores de automação residencial e industrial, transporte, segurança, controle ambiental e na saúde. Aplicações da área da saúde, por exemplo, podem monitorar o estado de saúde de pacientes utilizando informações do ambiente no qual o paciente se encontra (e.g. temperatura, pressão) e do paciente em si (e.g. frequência cardíaca, temperatura corporal, pressão sanguínea) e, a partir dessas informações, podem tomar medidas para ajudar no controle do estado do paciente, como fornecer informações sobre prescrição de medicamento ou alertar equipes médicas quando necessário.

O desenvolvimento de aplicações cientes de contexto e de IoT é uma tarefa complexa devido à natureza dinâmica e à heterogeneidade das fontes de informação e modelos de representação do conhecimento contextual. Informações de contexto podem ser extraídas a partir de sensores de baixo nível, mas também podem ser fornecidas por gerenciadores de alto nível ou serem derivadas de outras aplicações. A maioria das aplicações sensíveis ao contexto obtém inicialmente os dados a partir de sensores de baixo nível e posteriormente aplica sobre eles algum tipo de processamento para extrair algum conhecimento contextual. Podemos citar ainda como outros desafios relacionados ao desenvolvimento de aplicações cientes de contexto questões como coleta, processamento, armazenamento e distribuição das informações de contexto e a heterogeneidade de hardware (sensores, atuadores), protocolos de comunicação, modelos de representação, dentre outros aspectos.

Uma abordagem frequentemente explorada para mitigar a complexidade do desenvolvimento de aplicações cientes de contexto é o uso de *middleware* que escondem dos programadores os detalhes referentes ao gerenciamento do ciclo de vida das informações de contexto, à heterogeneidade de software e hardware, protocolos

e outros aspectos, fornecendo ao programador uma interface de programação (API - *Application Program Interface*) para desenvolvimento de aplicações cientes de contexto em um nível mais alto de abstração.

O *Laboratory for Advanced Collaboration* da Pontifícia Universidade Católica do Rio de Janeiro (LAC-PUC-Rio) e o Laboratório de Sistemas Distribuídos Inteligentes da Universidade Federal do Maranhão (LSDi-UFMA) têm, nos últimos anos, desenvolvido colaborativamente um *middleware* voltado ao desenvolvimento de aplicações cientes de contexto associado ao paradigma da IoT denominado M-Hub. O *middleware*, executado em dispositivos pessoais móveis, é responsável pela descoberta, conexão e aquisição de dados de baixo nível junto a objetos inteligentes (e.g. sensores, atuadores, relógios, robôs, veículos) próximos e acessíveis a partir de tecnologias WPAN (*Wireless Personal Area Network*) de curto alcance, como *Bluetooth Classic* e *Bluetooth Low Energy* (BLE), ou que estejam embutidos no próprio dispositivo móvel.

1.2 Motivação e Caracterização do Problema

Um aspecto importante que ganhou destaque nos últimos anos nas pesquisas relacionadas à computação ciente de contexto está relacionado à qualidade da informação de contexto consumida pelas aplicações. O termo Qualidade de Contexto (QoC - *Quality of Context*) possui diversas definições a depender do autor. Em uma visão restrita, expressa um conjunto de parâmetros que caracterizam apenas a Qualidade da Informação (QoI) que é usada como contexto (e.g. idade, acurácia) [15]. Em uma visão abrangente, a noção de QoC pode incluir também parâmetros que caracterizam a Qualidade do Serviço de distribuição da informação (QoS) (e.g. tempo de entrega dos dados, confiabilidade) [6].

A QoC é importante pois uma das principais características das informações de contexto é a imperfeição [7]. Tanto durante o processo de coleta das informações como durante seu processamento e distribuição, erros podem gerar informações de contexto imprecisas (por exemplo, um sensor mal calibrado). Tais imperfeições podem provocar erros na adaptação das aplicações sensíveis ao contexto. Por conta disso, além dos requisitos funcionais comumente definidos para uma aplicação, que expressam as informações de contexto que ela utilizará e quais serão os provedores dos serviços, é

necessário que sejam definidos os requisitos de qualidade da informação de contexto que a aplicação espera receber de modo a cumprir suas funcionalidades. Apesar de sua importância, a pesquisa sobre a qualidade do provisionamento de informações de contexto, que é a base para a construção de sistemas sensíveis ao contexto de alta qualidade, ainda é insuficiente [51].

As aplicações precisam estar cientes da QoC por diversos motivos. Por exemplo, a QoC pode ser usada como critério de consulta e seleção dos provedores de serviços. Assim, as aplicações podem buscar e escolher aqueles que melhor atendem seus requisitos de contexto e de QoC. A QoC também pode ser uma cláusula em acordos de nível de serviço (SLAs - *Service Level Agreements*), onde os produtores declaram os serviços de contexto e o nível de QoC que podem fornecer, enquanto que os consumidores especificam as informações de seu interesse e o nível de qualidade com a qual exigem recebê-las [15]. A QoC também pode ser usada para controlar a privacidade dos dados. Por exemplo, a acurácia da localização pode ser reduzida propositalmente em situações onde o usuário não deseja divulgar o lugar exato em que está [76].

Outro aspecto que devemos considerar no desenvolvimento de aplicações cientes de contexto é a grande quantidade de dados que podem ser gerados pelos diversos sensores presentes no ambiente. Em uma aplicação da área da saúde, por exemplo, o consumo de dados de batimentos cardíacos, frequência respiratória e outros dados do paciente, pode facilmente envolver grandes fluxos de dados que devem ser gerenciados pelas aplicações. Um mecanismo que pode ser fornecido por um *middleware* de desenvolvimento de aplicações cientes de contexto é a capacidade de se estabelecer filtros que diminuam este fluxo de informações de modo a reduzir o processamento dos dados e/ou o consumo da largura de banda na comunicação. Tal mecanismo pode utilizar a QoC como critério de filtragem e agir em dois momentos: a) no envio dos dados pelos sensores para as aplicações, onde o *middleware* poderia filtrar as informações de acordo com critérios de QoC de modo a publicar apenas aquelas que sejam de interesse de algum consumidor e b) no recebimento dos dados pelas aplicações, onde a aplicação utiliza o *middleware* para estabelecer um filtro que diminua o fluxo de dados de acordo com algum critério relacionado a parâmetros de QoC.

A oscilação na qualidade de contexto tem impacto direto sobre o funcionamento das aplicações e consequentemente na experiência de seus usuários

[16]. Em cenários da IoMT, a mobilidade dos dispositivos pode afetar a QoC, pois sensores com diferentes capacidades em relação à qualidade de contexto podem surgir e sumir repentinamente, alterando a oferta e o nível de QoC disponível no ambiente de execução. Portanto, a variação da QoC durante a execução de uma aplicação implica na necessidade de mecanismos de monitoramento que forneçam às aplicações notificações sobre mudanças que ocorrem na qualidade de contexto das informações durante a prestação do serviço. O monitoramento da QoC em tempo real é um passo fundamental em direção ao desenvolvimento de mecanismos de adaptação a variações na qualidade de contexto que permitam garantir o bom funcionamento das aplicações cientes de contexto, possibilitando que elas reajam não apenas em face das mudanças de contexto usuais, mas também diante das variações de QoC.

Sendo tão importante para as aplicações o conhecimento da qualidade das informações que utilizam, a capacidade de avaliar os valores dos parâmetro de QoC de uma informação de contexto se apresenta como fundamental para os *middleware* de desenvolvimento de aplicações cientes de contexto. Embora diversos *middleware* tenham sido propostos na literatura, poucas soluções apresentam suporte a QoC totalmente integrado [3]. Na literatura, algumas propostas se concentram apenas em QoS [47] enquanto outras apenas em QoI [42] e [29], dificultando o desenvolvimento de aplicações cujos requisitos de QoC abrangem QoI e QoS. Além disso, percebe-se que geralmente os *middleware* dão suporte a poucos parâmetros de QoC. O processo de avaliação de QoC apresenta alguns desafios:

- Não existe consenso na literatura quanto à nomenclatura ou à forma de avaliar determinados parâmetros de QoC [57]. Termos como acurácia, precisão, completude e outros têm significados diferentes de autor para autor. Uma abordagem que defina claramente os parâmetros de QoC suportados e como estes são avaliados permite às aplicações uma melhor utilização dessas informações.
- A heterogeneidade dos sensores implica em uma maior complexidade para o processo de avaliação pois não existe padrão nos modelos de representação da QoC dos diferentes tipos de sensores.

Apesar da importância do monitoramento da QoC, o avanço no desenvolvimento de mecanismos de monitoramento e adaptação dinâmica a variações

de QoC encontra obstáculos consideráveis impostos pelo estado da arte atual no que se refere ao suporte a QoC provido pelas soluções de *middleware* existentes. Isso porque, embora diversos *middleware* para desenvolvimento de aplicações cientes de contexto tenham sido propostos na literatura, poucas soluções apresentam suporte a monitoramento de QoC. O desenvolvimento de um mecanismo de monitoramento de QoC em sistemas cientes de contexto apresenta alguns desafios, dentre eles:

- A linguagem utilizada para expressar os requisitos de monitoramento: para o desenvolvedor a linguagem deve ser capaz de expressar vários requisitos de monitoramento e ser de fácil utilização. Como os dados a serem monitorados são heterogêneos e provenientes de múltiplos sensores, a linguagem que a aplicação utiliza deve ser uniforme e abstrair detalhes acerca dos tipos diversos de informações de contexto. O fato da qualidade de informação (QoI) e a qualidade do serviço de distribuição (QoS) serem monitoradas de forma diferente deve idealmente ser mitigado através de uma linguagem única e simples.

- Frequentemente as aplicações precisam alterar o monitoramento em tempo de execução. Regras devem poder ser adicionadas e removidas sem a necessidade de se reiniciar a aplicação.

- O impacto do processo de monitoramento no desempenho das aplicações. O monitoramento pode atuar em uma grande quantidade de dados que vêm dos provedores de informações de contexto. Se uma aplicação deseja monitorar varias informações de contexto durante um tempo considerável, o processo de monitoramento deve conseguir atuar sem afetar o desempenho da aplicação. De preferência, o monitoramento deve ser feito em um processo à parte da aplicação.

1.2.1 Objetivos

Objetivo Geral

Este trabalho de mestrado tem por objetivo geral investigar os problemas concernentes à avaliação de parâmetros de QoC, monitoramento de QoC e filtragem de informações baseadas em critérios de QoC e desenvolver mecanismos de

provisionamento de tais funcionalidades a serem incorporados numa abordagem de *middleware*.

Objetivos Específicos

- Levantar o estado da arte referente a *middleware* de contexto para aplicações da Internet das Coisas com suporte a avaliação, monitoramento e adaptação a variações de QoC.
- Desenvolver um mecanismo de avaliação da QoC considerando tanto a QoI como a QoS.
- Desenvolver um mecanismo de filtragem das informações de contexto que utilize a QoC como critério para o processamento dos filtros e que possa ser utilizado pelas aplicações para diminuir o fluxo de dados enviados ou recebidos.
- Projetar mecanismos de monitoramento e adaptação a variações de qualidade de contexto considerando os parâmetros e políticas de QoC providos pelo *middleware* M-Hub/CDDL.
- Avaliar os mecanismos de monitoramento e adaptação a variações de qualidade de contexto implementados.

1.3 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 fornece uma fundamentação teórica sobre Ciência de Contexto, Qualidade de Contexto, Processamento de Eventos Complexos e o *middleware* M-Hub. O Capítulo 3 descreve a solução proposta para a avaliação, filtragem e monitoramento de QoC. O Capítulo 4 apresenta a avaliação da proposta por meio de um experimento. O Capítulo 5 apresenta uma análise dos principais trabalhos relacionados e o Capítulo 6 mostra as conclusões desta proposta de dissertação de mestrado.

2 Fundamentação Teórica

2.1 Ciência de Contexto

A computação ciente de contexto refere-se à capacidade de um sistema computacional perceber características do meio ambiente que sejam de seu interesse e tomar decisões de acordo com mudanças nesse ambiente [73]. Tais aplicações reagem a ações executadas por outras entidades, podendo estas serem pessoas, objetos ou até mesmo outros sistemas, que modifiquem o ambiente. A computação ciente de contexto visa tornar a experiência do usuário em relação à utilização de recursos computacionais, mais agradável. Em cenários de computação ubíqua, o foco dos usuários seria a tarefa e não a ferramenta utilizada, e eles nem sequer perceberiam ou necessitariam de conhecimentos técnicos relativos aos recursos computacionais utilizados. Para que este nível de invisibilidade seja atingido, é fundamental que as aplicações sejam capazes de obter informações do ambiente no qual o usuário se encontra a fim de prover a ele serviços e informações úteis. Portanto, as aplicações ubíquas requerem ciência de contexto.

Segundo Schilit e Theimer [73], contexto indica onde o usuário está, com quem está e quais recursos estão disponíveis na vizinhança. Dey [28] considera como contexto qualquer informação que possa ser usada para caracterizar a situação de uma entidade (e.g., pessoas, objetos, *hardware*, o próprio sistema) que seja relevante para a interação entre o usuário e a aplicação, incluindo o usuário e a aplicação. Bolchini et al. [8] apresentam as informações de contexto como um conjunto de variáveis que podem interessar a uma entidade e influenciar suas ações.

As informações de contexto podem ser utilizadas para enriquecer a usabilidade das aplicações sensíveis ao contexto ao permitir que elas reajam a situações sem a necessidade de interação com o usuário. Por exemplo, um aplicativo pode reduzir o consumo de energia de um smartphone, caso detecte que o nível de bateria do sistema está baixo, reduzindo a luminosidade da tela ou desativando sensores que não estejam sendo utilizados.

2.1.1 Classificação de Contexto

Schilit e Theimer [73] propuseram uma classificação de contexto com três categorias: computacional, físico e do usuário. Chen e Kotz [19] adicionaram uma quarta categoria: o contexto temporal. Temos portanto a seguinte classificação:

- O **contexto computacional** engloba aspectos técnicos relacionados às capacidades dos recursos computacionais, tais como memória, processamento, níveis de energia, redes disponíveis e outros.
- O **contexto físico** aborda aspectos que representam o mundo real e que são acessíveis por sensores ou recursos dos equipamentos ao redor. Dados como localização, velocidade, nível de ruído, temperatura e outros são exemplos de informações de contexto físico. Devido à sua natureza, as informações de contexto físico são muito suscetíveis a erros devidos a falhas e imprecisões nos processos de coleta dos dados.
- O **contexto do usuário** compreende a dimensão social do usuário, tais como perfil, pessoas ao redor, situação social e outras.
- O **contexto temporal** captura a dimensão temporal dos eventos que ocorrem no ambiente. Tais informações podem ser, por exemplo, o dia da semana ou a hora em que determinada atividade acontece.

Emmanouilidis et al. [31] propõe uma classificação mais atual que apresenta cinco categorias de informações de contexto. A Figura 2.1 ilustra essa classificação. Fazendo um paralelo com a classificação anterior, podemos dizer que **Usuário** (*User*) equivale ao contexto do usuário, **Sistema** (*System*) equivale ao contexto computacional e **Ambiente** (*Environment*) equivale à união de contexto físico e contexto temporal. Outros dois tipos de contexto são propostos por essa classificação, são eles: **Social** e **Serviço** (*Service*).

O Contexto Social caracteriza as formas de relacionamento e de interações entre pessoas, intermediadas ou não por alguma tecnologia de comunicação. O conceito se refere ao ambiente social do usuário (no mundo real ou virtual) e à relação que pode ser estabelecida com outros usuários. Informações de contexto social podem ser extraídas por meio de redes sociais, sensores ou formulários e são aspectos de

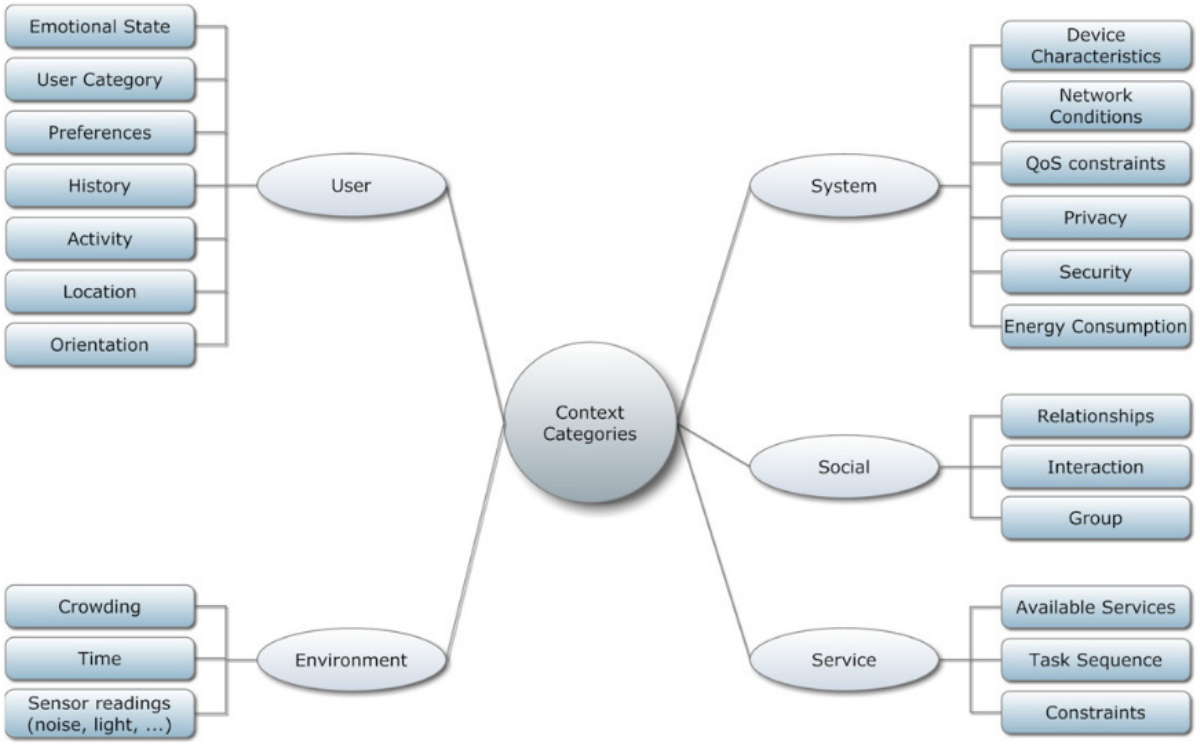

Figura 2.1: Classificação de Contexto segundo Emmanouilidis et al.

contexto de alto nível tais como o perfil do usuário, pessoas próximas, e sua atual situação social.

O Contexto de Serviço é qualquer informação que representa os serviços disponíveis no ambiente em que o usuário se encontra. Por exemplo, o nível de tráfego em uma rua (neste caso a rua é o serviço) ou o tempo de espera em uma fila de banco (neste caso o banco é o serviço). O contexto de serviço pode ser utilizado para guiar o usuário na realização de uma sequência de tarefas de acordo com o estado dos serviços, levando em consideração restrições (por exemplo, o horário de atendimento de determinado serviço ou a interdependência entre serviços).

2.1.2 Modelos de Representação de Contexto

As informações de contexto precisam ser representadas de uma maneira formal nos sistemas que as utilizam. Essa representação deve definir quais informações devem fazer parte do modelo e como elas são relacionadas. Um modelo de representação de contexto define os conceitos relevantes para o domínio de aplicação, os tipos de relacionamentos que o modelo abrange e o significado de cada relacionamento [27].

Existem diversas abordagens para modelagem de contexto em sistemas computacionais ubíquos propostas na literatura. Cada uma tem seus próprios custos de representação e processamento e limitações de expressividade.

Para Helf [41], uma abordagem para a representação de contexto abrangente deve ter as seguintes características:

- **Estruturação:** um modelo de representação bem estruturado facilita a filtragem e/ou extração das informações do contexto que são relevantes para a aplicação. Além disso, reduz a possibilidade de ambiguidade de atributos através do uso de identificadores únicos.
- **Intercambialidade:** capacidade de promover o intercâmbio de informações de contexto entre diferentes componentes do sistema sensível ao contexto.
- **Composição/Decomposição:** capacidade de compor ou decompor as informações de contexto. Considerando que uma informação de contexto pode ser composta por diferentes partes (atributos), essa característica ajuda na atualização da informação, já que isso possibilita transmitir apenas partes da informação, ao invés de enviar a informação completa a cada atualização.
- **Extensibilidade:** capacidade de adicionar ou remover atributos a fim de atender necessidades futuras de alteração no modelo existente.
- **Padronização**: capacidade de representar as informações utilizando um formato padrão que possa ser compreendido por diferentes entidades dos sistemas sensíveis ao contexto.

De acordo com Strang e Linnhoff-Popien [77], os modelos de representação de contexto podem ser agrupados nas abordagens descritas a seguir:

Par Chave-valor é considerada a abordagem a mais simples. A informação de contexto é modelada como uma tupla (chave, valor), ou seja, pares compostos por uma chave, que identifica o atributo de contexto, e por um valor associado a essa chave. Por exemplo, o contexto de saúde um paciente poderia ser representado pelas tuplas: (frequência cardíaca = 180bpm), (temperatura corporal = 37°C). Esse modelo não é recomendado para aplicações com estruturas complexas, pois não tem suporte

a hierarquias entre os atributos. Devido ao baixo nível de estruturação, não é possível utilizar-se algoritmos eficientes de recuperação das informações de contexto [64].

Esquema de Marcação é uma abordagem que representa as informações de contexto como uma estrutura hierárquica, na qual a especificação das informações é textual e delimitada por rótulos (*tags*) de marcação. A linguagem XML (*eXtended Markup Language*) é usada como padrão para a modelagem das informações de contexto, permitindo definir um vocabulário comum e extensível. Isso facilita o compartilhamento das informações de contexto por diferentes aplicações [80]. Uma desvantagem é que essa abordagem não consegue solucionar ambiguidades. Apesar de sua maior representatividade em relação ao esquema anterior, falta a esta abordagem capacidades importantes para descrever eventos complexos, como por exemplo a herança.

Orientada a Objetos é uma abordagem que utiliza o paradigma da Orientação a Objetos e seus conceitos como herança, encapsulamento e reuso. As informações de contexto são representadas através do uso de classes e atributos. Esta abordagem, ao contrário dos anteriores, possui a capacidade de validação dos dados como uma de suas características principais. Segundo [70], esse modelo não explora a descrição semântica das informações contextuais e, por isso, há uma incapacidade de implementação de algoritmos que realizem a inferência e interpretação de novos contextos a partir apenas da descrição destas informações.

Lógica é uma abordagem que utiliza a lógica formal para definir as condições em que uma expressão de conclusão ou fato pode ser derivada a partir de um conjunto de outras expressões ou fatos em um processo conhecido como raciocínio ou inferência. Para descrever essas condições em um conjunto de regras, um sistema formal é aplicado, onde o contexto é definido como fatos, expressões e regras. Os fatos podem ser adicionados manualmente na base, ou inferidos a partir de regras. Uma desvantagem dessa abordagem é a dificuldade para validar dados. Segundo Thomas e Linnhoff-Popien [78], modelos baseados em lógica possuem validação parcial, são difíceis de manter e muito suscetível a erros.

A abordagem baseada em **Ontologia** tira proveito da capacidade que ontologias têm de expressar relações mais complexas. Nessa abordagem é possível validar dos dados por meio da imposição de restrições da ontologia [6]. Ontologias

conseguem agregar conceitos e fatos. Com o uso de seus padrões é mais fácil promover o reuso e compartilhamento das informações de contexto, bem como eliminar ambiguidades [64]. Sistemas baseados em ontologias geralmente requerem maior capacidade de armazenamento e processamento.

2.1.3 Ciclo de Gerenciamento de Contexto

Existem diversas propostas de classificação das fases de gerenciamento de contexto. Neste trabalho, consideraremos a proposta por Bellavista et al. [6] que apresenta quatro fases principais no ciclo de gerenciamento de contexto: produção, processamento, armazenamento e distribuição.

Na fase de **produção**, as informações de contexto são coletadas do ambiente através de diversos processos e fontes. Sensores físicos podem nos fornecer informações de contextos físicos como temperatura ou nível de ruído. Informações de contexto temporais podem ser obtidas a partir de relógios ou temporizadores (tanto externos como internos em relação ao sistema). Informações de contexto do usuário podem ser coletadas em perfis ou por meio de associações com outros sistemas.

A fase de **processamento** engloba a realização de operações de transformação, tais como agregações ou filtragens, para atender às suas necessidades adaptativas do sistema. Agregação é o processo através do qual um sistema produz novas informações de contexto a partir de informações já coletadas de outras fontes. Por exemplo, as informações de localização e tempo, além da presença de outros usuários ao redor, podem ser combinadas para inferir que o paciente está no trabalho, o que constitui uma informação derivada. Filtragens são operações que visam reduzir o volume de dados que são recebidos do ambiente de modo a viabilizar o processamento eficiente do que realmente é necessário. Por exemplo, se para um determinado sistema a localização do usuário só é importante em determinado período do dia, então o sistema não precisa receber esta informação o dia todo, pois isto apenas acarretaria em processamento desnecessário.

A fase de **armazenamento** da informação de contexto é importante em casos em que o histórico da informação pode afetar o comportamento do sistema. Evidentemente, a capacidade de manter o histórico depende da capacidade dos recursos de armazenamento. Alguns sistemas precisam apenas da informação atual,

mas que deve permanecer acessível durante um determinado tempo apenas, o tempo de validade da informação. Em outros casos, todo o histórico da informação de contexto é necessário para o correto funcionamento do sistema. Como exemplo do segundo caso, podemos citar registros dos sinais vitais do paciente que são importantes para a composição de seu histórico médico.

Por fim, temos a fase de **distribuição** de informações de contexto. Esta fase está presente em sistemas que enviam informações do contexto para outros sistemas interessados nas mesmas. Normalmente os gerenciadores de contexto são utilizados para desenvolvimento de sistemas cientes do contexto possuem componentes especializados na distribuição dos dados aos interessados.

Perera et al. [68] promovem uma análise dos ciclos de vida de informação de contexto mais recorrentes na literatura, e com base nessa análise propõe um ciclo de vida semelhante ao anterior, contendo as seguintes fases: aquisição, modelagem, raciocínio e disseminação. Nessa proposta, em que o processamento ocorre em duas fases (modelagem e raciocínio), o armazenamento não é considerado uma fase essencial. Já as fases de aquisição e disseminação correspondem respectivamente as fases de produção e distribuição da proposta anterior.

2.1.4 Modelos de Distribuição de Informações de Contexto

Em relação a distribuição de dados de contexto, existem vários modelos de comunicação que permitem a entrega das informações produzidas aos interessados. Geralmente os modelos de comunicação se diferenciam em relação ao acoplamento entre produtores e consumidores e à sincronia na comunicação entre eles. Os principais modelos empregados são: passagem de mensagem, publicador/subscritor e espaço de tuplas.

Modelos de distribuição baseados em passagem de mensagens possuem a desvantagem de acoplamento forte entre os participantes. Neste modelo, é necessário que o emissor tenha conhecimento da identidade exata e do endereço do receptor. Além disso, existe a necessidade de sincronização, isto é, o emissor precisa esperar até que o receptor esteja pronto para trocar de informações. Em sistemas sensíveis ao contexto, esta característica é bastante restritiva pois nem todas as aplicações ubíquas requerem modelos onde produtores e consumidores de informações estejam ativos

ao mesmo tempo. Dessa forma, torna-se necessária a utilização de um modelo de comunicação mais desacoplado, e que permita produtores e consumidores utilizarem em tempo diferentes, independentemente de eventuais desconexões.

O modelo publicador/subscritor (*publish/subscribe*) oferece um mecanismo assíncrono para troca de mensagens. Nesse modelo as mensagens são publicadas pelos produtores (*publishers*) e são entregues automaticamente a todos os consumidores (*subscribers*) inscritos para recebê-las. Esse modelo desacopla produtores e consumidores, sendo mais adequado para distribuição de dados de contexto em ambientes móveis e cenários de larga escala [36].

Outro modelo de comunicação que evita o problema de forte acoplamento em sistemas sensíveis ao contexto é o baseado em espaços de tuplas. Um espaço de tuplas é espaço de memória compartilhado globalmente e distribuído por todos os processos e *hosts* participantes. Os processos que usam este modelo se comunicam gerando tuplas que são enviados ao espaço de tuplas. Tuplas são dados estruturados tipados, como por exemplo, objetos em C++ e Java, onde cada tupla é formada por uma coleção de campos de dados tipados e representa uma parte coesa da informação. Em um sistema baseado em espaço de tuplas, todas as comunicações entre produtores e consumidores são exclusivamente realizadas utilizando este espaço. Os produtores inserem as tuplas no espaço compartilhado enquanto que os consumidores leem as tuplas neste espaço [36].

2.2 Qualidade de Contexto

Os serviços de distribuição de contexto disponibilizam uma infraestrutura de suporte para coleta, gerenciamento e distribuição das informações de contexto sobre uma série de temas, que podem estar relacionados ao usuário, a objetos ou até mesmo ao ambiente. Tais serviços adquirem informações de contexto de várias fontes. Muitos consumidores de informações de contexto, como aplicações voltadas para a área da saúde, requerem que os dados sejam entregues considerando diversos aspectos de qualidade, sendo este um pré-requisito para que os mesmos possam atingir os seus objetivos.

2.2.1 Definições de QoC

Buchholz et al. [15] definem QoC como sendo qualquer informação que descreve a qualidade da informação que é usada como informação de contexto. Krause e Hochstatter [46] adicionaram o conceito de valor à definição de QoC para introduzir o ponto de vista da aplicação alvo. Para eles, enquanto a QoC é algo inerente à informação e pode ser estabelecida de maneira objetiva e independente de sua utilização, a estimativa do quão valiosa é a informação (o valor) depende das características e necessidades do consumidor em relação à informação, sendo algo mais subjetivo. Com base nesta distinção entre qualidade e valor, estes autores definiram QoC como sendo qualquer informação inerente que descreve uma informação de contexto e que pode ser utilizada para determinar o valor que a informação possui para uma aplicação específica. Manzoor et al. [56,58,59] propuseram dividir a noção de QoC em duas visões: a objetiva e a subjetiva. A visão objetiva é independente da situação em que as informações de contexto são utilizadas e das exigências dos consumidores. Essa visão objetiva de QoC é determinada pelas características dos sensores utilizados para coletar os dados de contexto, bem como pelas condições em que a medição do valor de contexto foi feita. A visão subjetiva expressa o quanto uma informação de contexto está em conformidade com as exigências de uma aplicação consumidora em particular. Considerando as visões objetiva e subjetiva de QoC, eles definiram QoC como sendo uma indicação do grau de conformidade das informações de contexto coletadas por sensores em relação à situação que prevalece no ambiente e às exigências de um determinado consumidor de contexto.

A qualidade de contexto pode ser utilizada para diferentes propósitos. Com base nos trabalhos de Buchholz et al. [15] e Sheikh et al. [76], identifica-se algumas razões para se considerar explicitamente o uso de QoC: estabelecimento de contratos de QoC entre publicadores e consumidores, seleção de publicadores de contexto apropriados baseando-se na QoC oferecida, adaptação do serviço de distribuição de contexto, melhoria da experiência do usuários e privacidade do usuário.

2.2.2 Parâmetros de QoC

As informações de contexto são coletadas de diversas fontes. Podem, por exemplo, serem coletadas de sensores no ambiente ou recebidas de usuários. Os parâmetros de QoC são informações associadas aos dados de contexto que têm o propósito de identificar a qualidade destes dados. Os valores dos parâmetros de QoC indicam o quão um dado de contexto está próximo do valor ideal para atender à necessidade de determinado cliente.

Abordamos as nomenclaturas e os conceitos que definem a semântica de diversos parâmetros de QoC encontrados na literatura, tomando como ponto de partida as análises de trabalhos que propuseram parâmetros de QoC promovidas por Chabridon et al. [18] e Nazario [65]. Assim como ocorre em relação as definições de QoC, não existe uma uniformização em relação as nomenclaturas e semânticas dos parâmetros de QoC. Algumas vezes, autores diferentes usam nomenclaturas diferentes para parâmetros que tem a mesma semântica. Em outras, eles usam a mesma nomenclatura para parâmetros que possuem significados diferentes.

Apresentamos a seguir definições dos principais parâmetros de QoC. Para tal, dividimos os parâmetros em duas categorias. A primeira categoria representa os parâmetros de QoC que dizem respeito às informações de contexto em si e aos sensores que as coletam (se referem à qualidade dos dados entregues à aplicação consumidora). A segunda categoria representa os parâmetros de QoC que se referem ao serviço de distribuição das informações (expressam a qualidade da entrega dos dados e determinam para o serviço de distribuição onde, como e quando entregar as informações).

2.2.3 Parâmetros de Qualidade das Informações/Sensores

- **Acurácia** - Em Metrologia, de acordo com Marquis et al. [63], o termo acurácia representa o quão próximo um valor mensurado está do valor real. Em Ciência de Contexto, Filho [34], Manzoor et al. [59], e Preuveneers et al. [72] concordam com essa definição. Buchholz et al. [15] utilizam limites para representar o desvio do valor mensurado em relação ao valor exato. Kim e Lee [45] estimam precisão usando intervalos de confiança obtidos por um método estatístico clássico.

- **Atributos Disponíveis** - Manzoor et al. [58, 59] define o parâmetro atributos disponíveis como a quantidade de atributos que são disponibilizados para uma determinada informação. Por exemplo, informações de localização coletadas por um GPS podem ser disponibilizadas com três atributos: latitude, longitude e altitude.

- **Atributos Requeridos** - Para Manzoor et al. [58, 59], o parâmetro atributos requeridos corresponde ao número de atributos de contexto que têm um valor significativo para uma aplicação consumidora específica.

- **Atualidade** - Manzoor et al. [58,59] definem o conceito de atualidade como sendo o grau de racionalismo, que aqui pode ser entendido como o grau de perfeição ou validade, em relação ao uso de uma informação de contexto, normalizado no intervalo de zero a um ($[0..1]$). Em outras palavras, esse autor calcula a atualidade da informação de contexto com base na razão entre a idade da informação de contexto e um dado tempo, que pode ser intervalo entre as medições (na visão objetiva) ou tempo de validade (na visão subjetiva). Quanto menor o valor de atualidade, maior é a probabilidade de a informação ser enganosa ou conter erros ou não ser mais válida para o sistema sensível ao contexto, podendo ser descartada.

- **Completude** - A completude ou integralidade indica a quantidade de atributos de contexto que são providos [45]. Manzoor et al. [58, 59] associam um peso para cada atributo de contexto para representar a sua importância e calculam a completude como base na razão entre a soma dos pesos dos atributos disponíveis e a soma dos pesos do número total de atributos requeridos. Han et al. [40] utilizam o termo confiabilidade como sinônimo de completude, e a definem como sendo o número mínimo de dados do sensor que devem ser coletados dentro de alguma unidade de tempo.

- **Confiança** - Filho [34] identifica duas abordagens para medir a confiança: (i) medir o quão confiável é a entidade que forneceu a informação de contexto, tal como proposto por Buchholz et al. [15] e; (ii) medir a confiança que alguém possa atribuir diretamente a informação de contexto. Na primeira abordagem, Huebscher e McCann [43] propõem um modelo de aprendizagem que calcula a confiança baseado em um *feedback* binário (positivo/negativo) dos usuários.

Manzoor et al. [58] segue a segunda abordagem, sendo que posteriormente em [59] passaram a usar a nomenclatura confiabilidade para expressar confiança, propondo calculá-la com base na acurácia e no inverso da distância entre o sensor e a entidade sobre a qual a informação de contexto é coletada. De acordo com esse cálculo, quanto mais próximo o sensor estiver da entidade da qual se quer coletar informações de contexto, maior será a confiança na informação obtida. Neisse et al. [66] argumentam que a confiabilidade é relacionada à capacidade do provedor de contexto de descrever de forma confiável os níveis de QoC que ele pode garantir. Mesmo assim, é possível que o provedor esteja fornecendo informações de contexto com níveis de QoC abaixo do que está sendo anunciado.

- **Distância do Sensor** - Manzoor et al. [59] define o parâmetro distância do sensor como a distância máxima com a qual o sensor consegue capturar a informação de contexto a respeito de uma entidade. Por exemplo, câmeras profissionais conseguem obter imagens a diversos metros de distância do local do alvo. Já um telescópio consegue imagens dos astros mesmo estando a milhares de quilômetros deles.

- **Idade** - Manzoor et al. [59] definem o parâmetro idade como sendo o tempo transcorrido entre o instante da medição da informação de contexto no sensor e o tempo de entrega ao consumidor ou o tempo atual. Sendo assim, a idade pode ser considerada um parâmetro dinâmico, porque aumenta a medida que o tempo passa. Buchholz et al. [15] utilizam a nomenclatura *up-to-dateness*, enquanto que Kim e Lee [45] e Sheikh et al. [75] utilizam a nomenclatura *freshness* para esse mesmo conceito. A idade da informação pode ser utilizada para determinar o quão atual ela é.

- **Localização da Fonte** - Manzoor et al. [59] define o parâmetro localização da fonte como o local onde a informação de contexto foi medida.

- **Instante da Medição** - Manzoor et al. [59] define o instante da medição como sendo o momento em que a informação de contexto foi medida.

- **Precisão** - Em Metrologia, de acordo com Marquis et al. [63], a precisão pode ser compreendida como sendo o grau de dispersão de um conjunto de medidas do mesmo sensor, independente do valor real. Em Ciência de Contexto, Buchholz

et al. [15] utilizam o termo precisão como sinônimo de acurácia. Filho [34] define precisão como o nível de detalhamento com que a informação de contexto descreve uma entidade do mundo real. Para ele a precisão pode ser avaliada a partir do cálculo da razão (divisão) entre o nível da precisão corrente e o nível de precisão máximo.

- **Privacidade** - Privacidade é um parâmetro que especifica o grau de publicidade de uma determinada informação. Filho e Agoulmine [33] utilizam o termo sensibilidade como sinônimo de privacidade. O nível de privacidade pode ser alterado pelos proprietários do contexto de acordo com a situação em que eles se encontram.

- **Probabilidade de Correção** - A probabilidade de correção ou exatidão estima o quão frequentemente uma informação de contexto está involuntariamente errada devido a erros internos [15, 71, 75]. Brgulja et al. [10] propõem a combinação de diferentes parâmetros de QoC como atualidade, confiabilidade e precisão da fonte de contexto para calcular a probabilidade de correção das informação de contexto. Filho e Agoulmine [33] estenderam esse trabalho, levando em conta dependências entre os dados de contexto que podem afirmar ou contradizer simultaneamente a verdade sobre caracterização de uma dada situação. Consequentemente, uma elevada probabilidade de correção indica que são pequenas as chances do contexto associado estar em contradição com outra informação de contexto.

- **Resolução** - Para Buchholz et al. [15], o parâmetro resolução, chamado por Manzoor et. al [59] de **granularidade**, indica a o grau de detalhes com o qual um sensor pode coletar dados, e faz parte das características do sensor. Filho [34] calcula a resolução como base na razão entre o nível de granularidade corrente e o nível máximo de granularidade. Sheikh et al. [75] apresentam dois tipos de resolução: a temporal, também conhecida como tempo de validade, e a espacial. Ambas são descritas a seguir.

- **Resolução Temporal** - Segundo Sheikh et al. [75] a resolução temporal é o período de tempo para o qual uma única amostra da informação de contexto é aplicável ou válida. Pra esse autor, o tempo de validade não pode ser superior ao intervalo entre as medições do sensor. Por exemplo, se a temperatura da sala é medida

pelo sensor ambiental de oito em oito horas, então esse é o tempo máximo pelo qual essa informação é considerada válida. Gray e Salber [39] utilizam o termo repetibilidade como sinônimo de resolução temporal. Manzoor et al. [59] utilizam o termo tempo de validade para se referir ao conceito de resolução temporal apresentado. No entanto, eles observam que o tempo de validade é um requisito do consumidor, e que independe das características ou configurações do sensor, sendo algo subjetivo, que varia de acordo com o dado de contexto e entidade que está sendo observada. Por exemplo, a localização de uma pessoa andando de carro em alta velocidade é valida por menos tempo que a localização de uma pessoa andando.

- **Resolução Espacial** - Segundo Sheikh et al. [75] a resolução espacial é a precisão (em termos de granularidade) com que é expressa a área física na qual a informação de contexto é aplicável. A resolução espacial é utilizada comumente para fins de privacidade. Por exemplo, um sistema de segurança pode saber quantas pessoas estão em determinado prédio mas não saber suas localizações exatas. Isto é possível porque os usuários podem informar sua localização com uma resolução espacial menos precisa. A quantificação deste parâmetro depende de como o "espaço" é definido pela aplicação. Para localizações de um GPS, por exemplo, a resolução espacial pode ser representada como um raio ao redor das coordenadas. Já o sistema de segurança citado pode obter os dados com uma resolução espacial de uma sala, um andar ou um edifício.

- **Significância** - O parâmetro significância foi proposto por Manzoor et al. [58, 59] para representar o valor ou preciosidade de uma informação de contexto para uma aplicação específica. Foi considerada também por Filho [34], que assim como Manzoor avalia a significância como base no resultado do cálculo da razão entre o valor crítico da informação de contexto e o valor crítico máximo que essa informação poderia ter.

- **Tempo de Validade** - Manzoor et al. [59] define o parâmetro tempo de validade como o período pelo qual a informação de contexto permanecerá útil para o consumidor. O próprio consumidor especifica esse parâmero.

- **Usabilidade** - Para Manzoor et al. [58, 59] parâmetro usabilidade descreve o quanto um pedaço de informação de contexto é adequado para ser usado com

o propósito a que se destina pela aplicação consumidora. O valor de usabilidade é igual a 1 (um) quando o nível de granularidade da informação de contexto coletada é maior do que o nível de granularidade necessário. Caso contrário o valor de usabilidade é 0 (zero).

- **Valor Crítico** - Para Manzoor et al. [59] o parâmetro valor crítico corresponde ao nível de importância da informação de contexto para uma aplicação específica.

2.2.4 Parâmetros de Qualidade do Serviço de Distribuição

- **Confiabilidade na Entrega** - A confiabilidade na entrega é o parâmetro que determina se o serviço de entrega deve utilizar mecanismos de detecção e retransmissão em caso de perda de mensagens. É utilizado por diversos serviços de distribuição de dados como DDS (*Data Distribution Service*) [67] e MQTT (*Message Queue Telemetry Transport*) [12].

- **Durabilidade** - A durabilidade permite que os publicadores de contexto especifiquem mensagens persistentes que serão armazenadas e entregues posteriormente.

- **Frequência** - Para Gray e Salber [39] o parâmetro frequência equivale a taxa de amostragem do sensor (em hertz), ou seja, é o resultado obtido a partir da razão entre um determinado número de medições realizadas pelo sensor e um dado intervalo de tempo. Essa frequência também pode ser definida com base em um intervalo fixo entre medições sucessivas. Manzoor et al. [59] chamam esse intervalo simplesmente de período de tempo (*time period*).

- **Histórico** - O histórico permite que publicadores e consumidores definam o tamanho do histórico de mensagens enviadas e recebidas, respectivamente.

- **Intervalo de Medição** - Manzoor et al. [59] define o parâmetro intervalo de medição como o tempo entre medições sucessivas do sensor. Em algumas situações, o usuário do sensor pode ajustar esse parâmetro. Em outras, o sensor já possui um intervalo pré-definido, que não pode ser alterado pelo usuário.

- **Ordem de Destino** - Segundo Pardo-Castellote [67], o parâmetro ordem de destino especifica se as informações inseridas no histórico devem ser organizadas

com base no tempo em que a informação foi enviada pelo publicador ou com base no tempo de chegada do dado ao subscritor. Ao considerar o tempo do publicador é necessário que haja um mecanismo de sincronização entre os relógios do publicador e do subscritor.

- **Prioridade de Transporte** - Para Corradi et al. [23], a prioridade visa permitir o tráfego diferenciado quando múltiplos dados precisam ser encaminhados.

- **Taxa de Atualização** - Para Huebscher e McCann [43] o parâmetro taxa de atualização está relacionado com a idade da informação de contexto e descreve a frequência ou o intervalo de tempo com a qual as aplicações consumidoras podem ou gostariam de receber novas amostras de determinadas informações de contexto, independentemente da frequência com a qual elas são produzidas. Diferente da frequência de medição e do período de tempo que são configurados no sensor, a taxa de atualização é configurada no serviço de distribuição de dados de contexto.

- **Tempo de Atraso** - Para Bu et al. [14], o tempo de atraso é o intervalo de tempo entre o momento em que a situação acontece no mundo real e o momento em que a situação é reconhecida em computadores.

- **Tempo de Recuperação dos Dados** - Para Corradi et al. [23] o tempo de recuperação dados equivale ao intervalo de tempo entre a geração da requisição de uma informação de contexto e a entrega da resposta contendo o dado solicitado. Esse parâmetro expressa o tempo máximo que o consumidor de contexto está disposto a esperar por uma informação específica. Após esse intervalo de tempo, a resposta de contexto será inútil.

- **Vida Útil** - Segundo Pardo-Castellote [67], o parâmetro vida útil especifica o tempo pelo qual o serviço manterá as informações no histórico. Quando o prazo expira, as informações são deletadas.

- **Vivacidade** - Segundo Pardo-Castellote [67] o parâmero vivacidade indica se o serviço de distribuição deve tornar os subscritores cientes de falhas nos publicadores.

2.2.5 Motivação para o uso da QoC

Os trabalhos de Buchholz et al. [15] e Sheikh et al. [76] mostram algumas razões para se considerar explicitamente o uso de QoC. São elas:

1. **Estabelecer Contratos de QoC**: a QoC pode ser usada para firmar, quando necessário, um contrato de qualidade entre produtores e consumidores. Por exemplo, se a aplicação deseja receber dados de localização a cada 30 segundos, mas o produtor só pode enviá-los a cada 60 segundos, então um contrato não é fechado. Logo, o o serviço não será contratado.

2. **Selecionar Provedores de Contexto**: ainda que seleção do provedor de contexto não dependa de um contrato, o consumidor pode escolher aquele cuja qualidade de contexto atende melhor aos seus requisitos.

3. **Aumentar a Eficiência**: para aumentar a eficiência na utilização da largura de banda, o middleware de contexto poderá armazenar informações de contexto na *cache* do consumidor. Por exemplo, dependendo do intervalo entre as medições das informações, as aplicações consumidoras poderão obtê-las a partir da *cache* local ao invés de solicitar o envio de um novo dado ao provedor.

4. **Melhorar a Qualidade de Experiência do Usuário**: considerando que as informações de contexto são utilizadas para adaptar o conteúdo e os serviços providos pelas aplicações sensíveis ao contexto, é certo que a qualidade de contexto tem impacto direto sobre o funcionamento dessas aplicações e consequentemente na experiência de seus usuários. Se a experiência dos usuários melhora na medida em que a qualidade das informações de contexto aumenta, é preciso buscar mecanismos que mantenham a QoC em níveis suficientes para que os usuários das aplicações sensíveis ao contexto fiquem satisfeitos com seus serviços.

5. **Prover Privacidade**: existe uma relação entre a privacidade e a QoC. O *middleware* precisa fornecer aos usuários meios para limitar a QoC de suas informações pessoais. Por exemplo, a localização do usuário pode ter a acurácia reduzida propositalmente em situações em que ele não deseja ser encontrado.

2.2.6 Fatores que degradam a Qualidade de Contexto

Diversos fatores podem degradar a qualidade de contexto. Por exemplo, os sensores físicos, devido às suas próprias naturezas, podem estar sujeitos a erros de leitura provocados por má calibração ou falha de fabricação. Os sensores também podem estar mal posicionados e gerar informações incorretas. Para informações de contexto baseadas em perfis do usuário, a omissão do usuário em fornecer entradas pode fazer com que algumas informações sejam desconhecidas ou incompletas.

Alguns fatores que degradam a qualidade do serviço de distribuição:

1. **Falhas na Comunicação**: perda de pacotes, atrasos (*delay*) e variações de atrasos (*jitter*), conectividade fraca ou intermitente, desconexões, *handovers* de nós móveis:

2. **Falhas em componentes de hardware**: rompimento de cabos, defeitos em antenas ou satélites, quedas de energia que ocasionam o desligamento de servidores e estações de trabalho, problemas em estações rádio base utilizadas em redes móveis;

3. **Falhas em componentes de software**: travamento de aplicações, erros de lógica de programação que levam a exceções não previstas, falhas do sistema operacional ou na própria aplicação;

4. **Limitações de escalabilidade**: situações em que a quantidade de usuários, o grande volume de dados transmitidos e alocação ineficiente dos recursos de processamento, armazenamento e comunicação provocam congestionamentos e aumento do tempo de resposta ao consumidor.

Problemas inerentes as fases de aquisição podem influenciar, mesmo que indiretamente, no tempo de entrega das informações ao consumidor. Por exemplo, falhas nos sensores ou longos intervalos entre medições podem fazer com que a informação de contexto demore a ser enviada. Consequentemente haverá atrasos e as taxas de atualização contratadas deixarão de ser cumpridas.

2.3 Monitoramento da QoC

Uma característica da QoC é ela é dinamicamente, variável, ou seja, ela oscila ao longo do tempo. No melhor cenário, a QoC do provedor de serviço selecionado irá melhorar, ficando acima das expectativas da aplicação. No pior caso, essa QoC cairá de tal forma que, em algum momento, não mais atenderá os requisitos inicialmente especificados pela aplicação. Diversos fatores podem degradar a QoC em tempo de execução, tais como falhas de hardware e software dos sensores, gateways, servidores, rede de comunicação, dentre outras. A localização física dos sensores também é um fator que pode afetar a qualidade. O sensor pode estar mais ou menos próximo da fonte da informação, Condições como temperatura do ambiente, clima, pressão também devem ser considerados.

Há varias razões pelas quais aplicações desejam estar cientes do nível de QoC disponível. Por exemplo, a QoC pode ser usada como critério de consulta e seleção dos provedores de serviços. Assim, as aplicações podem buscar e escolher aqueles que melhor atendem seus requisitos de contexto e de QoC ao mesmo tempo. A QoC também pode ser uma cláusula a ser definida em acordos de nível de serviço (SLAs - *Service Level Agreements*), onde os produtores declaram os serviços de contexto e o nível de QoC que podem prover, enquanto que os consumidores especificam as informações de seu interesse e o nível de qualidade com a qual exigem recebê-las. A decisão de usar ou não determinado serviço pode ser realizada em função da qualidade das informações de contexto ou da qualidade da distribuição dos serviços.

Desse modo, o monitoramento da QoC é uma característica importante de sistemas que utilizem informações de contexto para a tomada de decisão. Durante o funcionamento do sistema que consume as informações de contexto, é necessário que o mesmo tenha a possibilidade de averiguar se as informações continuam atendendo aos seus requisitos de QoC. Como os provedores das informações estão sujeitos a falhas ou diminuição da QoC das informações que fornecem, o monitoramento dessa qualidade se torna crucial. Com o monitoramento, as aplicações podem tomar medidas adaptativas como substituir um provedor que não mais atende seus requisitos por outro provedor. Por exemplo, em um sistema de saúde, uma aplicação que esteja monitorando um paciente pode exigir que as informações do mesmo cheguem em intervalos de tempo pré-determinados. Caso a QoC do provedor caia e ele não consiga

mais fornecer as informações no intervalo acordado, a aplicação consumidora, através do monitoramento, fica ciente deste fato e pode trocar de provedor ou tomar outra medida.

Para exemplificar melhor a questão do monitoramento, consideremos um exemplo hipotético de uma aplicação ciente de contexto que monitore a localização de pacientes utilizando o sensor de um *smartphone* com o sistema operacional *Android*. Ao desenvolver uma aplicação *Android* ciente da localização, existem duas APIs que podem ser utilizadas: provedor de GPS ou provedor de localização de rede. A localização baseada em GPS possui maior acurácia, no entanto consume mais bateria e tem um atraso no envio da informação. Já a localização baseada na rede, usa torres de celulares e sinais de Wi-Fi para fornecer a informação e funciona tanto dentro como fora de ambientes. A resposta ao usuário é mais rápida e usa menos bateria. Uma aplicação que precise obter a localização do usuário pode ter como requisito uma acurácia maior e decidir usar um provedor de GPS. A aplicação pode decidir então monitorar a qualidade da informação para, caso o sinal do GPS se deteriore (por conta, por exemplo, do usuário estar em ambiente fechado), ela possa ser avisada e ter a opção de alternar para um sensor baseado em rede. Nesse caso, o monitoramento do provimento do sinal de GPS pode continuar para que a aplicação possa ser avisada quando este for restabelecido e ela volte a a usar o sinal de maior acurácia.

A confiabilidade de uma informação de contexto é um outro parâmetro muito importante para ser monitorado em sistemas cientes de contexto. Considere por exemplo, um sistema que monitore sinais vitais de um paciente com alguma doença crônica.Nesse caso, se a informação não for confiável, ela não tem utilidade pois deve ser usada para a tomada de decisões criticas. Se a frequência cardíaca passa a ser informada com um grau de confiabilidade menor do que o requisitado quando a aplicação solicitou o serviço ao provedor, esta tem que ser informada para que possa tomar as providências necessárias. Para que seja possível esta notificação, um serviço de monitoramento deve estar ativo durante todo o provimento do serviço. O monitoramento poderá então alertar a aplicação quando a confiabilidade estiver abaixo de um determinado valor previamente acordado.

Por fim, podemos citar o parâmetro de tempo de validade da informação de contexto. Uma aplicação, ao solicitar a um provedor uma informação pode especificar o tempo de validade do dado. Tal informação pode não se recebida após decorrido

este tempo. Por exemplo, uma aplicação que monitore os batimentos cardíacos de um paciente pode ter como requisito receber a informação com um tempo de validade de 1 segundo. Amostras que cheguem à aplicação e sejam mais velhas que este valor não servem. Nesse caso, um serviço de monitoramento de QoC pode ser configurado pela aplicação para avisá-la se, digamos, 3 amostras forem entregues seguidamente com um tempo de validade expirado. Note-se que em sistemas como os exemplificados anteriormente o monitoramento da QoC é uma funcionalidade imprescindível.

Como dito anteriormente, o desenvolvimento de mecanismos de monitoramento e adaptação dinâmica a variações de QoC encontra obstáculos consideráveis impostos pelo estado da arte atual no que se refere ao suporte a QoC provido pelas soluções de middleware existentes. Poucas soluções existentes apresentam suporte a QoC totalmente integrado com o monitoramento. Além disso, geralmente as soluções apresentadas estão focadas em uma outra dimensões de qualidade, dificultando o desenvolvimento de aplicações cujos requisitos de QoC abrangem QoI e QoS. Se o provisionamento da qualidade de contexto não é satisfatório, então monitorar a QoC para adaptar-se às suas variações torna-se inviável.

2.4 Processamento de Eventos Complexos

Com o número cada vez maior de redes de sensores e dispositivos inteligentes coletando interruptamente uma grande quantidade de dados, o problema de se analisar um fluxo cada vez maior de dados em tempo quase real se torna crítico. Diversos negócios têm como requisito importante a capacidade de sistemas reagirem rapidamente às mudanças no ambiente detectadas por sensores que coletam estes dados e esta capacidade pode ser um fator decisivo para o sucesso ou o fracasso de uma empresa. Um problema-chave no processamento em tempo real é a detecção de padrões de eventos em fluxos de dados.

O processamento de eventos complexos (CEP - *Complex Event Processing*) [54] aborda exatamente este problema de fazer a correlação de eventos de entrada contínuos e padrões de interesse, onde o resultado podem ser outros eventos complexos que são derivados dos eventos de entrada. CEP é um método de rastreamento e análise de baixa latência (de processamento) de fluxos de informações

(dados) que combina dados de múltiplas fontes para inferir eventos ou padrões que sugerem circunstâncias mais complicadas [38]. O processamento de eventos complexo é usado para coletar, processar, analisar fluxos de dados em tempo real e produzir resultados sem atrasos mesmo em caso de alto fluxo de eventos sendo recebidos. Em uma inversão em relação aos sistemas gerenciadores de bancos de dados tradicionais, onde uma consulta é executada em dados armazenados, o CEP executa dados em uma consulta armazenada. Todos os dados que não são relevantes para a consulta podem ser descartados imediatamente. As vantagens desta abordagem são óbvias, uma vez que as consultas CEP são aplicadas em um fluxo potencialmente infinito de dados. Além disso, as entradas são processadas imediatamente, o que conduz efetivamente à capacidade de análise em tempo real do CEP. Ele permite que se detecte situações em uma séries de eventos (que podem ser históricos ou em tempo real) e ações personalizadas sejam tomadas quando ocorrerem condições previamente definidas.

O CEP pode ser aplicado em uma ampla variedade situações. Por exemplo, CEP é usado hoje em dia em aplicações financeiras, como tendência do mercado de ações e detecção de fraude de cartão de crédito. Uma outra aplicação é o monitoramento e rastreamento baseado em RFID, por exemplo, para detecção de furtos em um armazém. O CEP também pode ser usado para detectar invasão de rede especificando padrões de comportamento suspeito do usuário.

2.4.1 Agentes de Processamento de Eventos e Redes de Processamento de Eventos

Uma rede de processamento de eventos (EPN - *Event Processing Network*) consiste em quatro componentes: Produtor de Eventos, Consumidor de Eventos, Agente de Processamento de Eventos (EPA - *Agent Processing Agent*) e um componente de conexão chamado Canal de Eventos. Uma EPN descreve como os eventos recebidos dos produtores são direcionados aos consumidores através de agentes que processam esses eventos realizando transformações, validações ou enriquecimento. Os eventos vão de um componente para outro através do Canal de Eventos. A Figura 2.2 apresenta um exemplo de rede de processamento de eventos com todos os componentes. Nela podemos perceber que a EPN pode utilizar alguma forma de armazenamento para persistir o estado de determinadas informações.

- **Canal de Eventos**: Um canal de eventos é um mecanismo para distribuir fluxos de eventos de produtores e agentes de processamento de eventos para consumidores e agentes de processamento de eventos. Ele está associado a um tipo de evento único, significando que todos os eventos que são transferidos através deste canal têm a mesma estrutura e são instâncias da mesma classe de eventos.

- **Produtor de Eventos**: É a fonte ou emissor de eventos, produz e publica eventos através de canais de eventos para os interessados (consumidores de eventos ou agentes de processamento de eventos).

- **Consumidor de Eventos**: É o componente interessado nos eventos de modo a executar suas atividades. Ao receber um evento, o consumidor pode realizar uma determinada tarefa associada a este evento.

- **Agente de Processamento de Eventos**: Em um sistema distribuído e heterogêneo, os produtores podem produzir os eventos de uma maneira diferente da qual os consumidores espera receber. Esses eventos podem ter estrutura ou significado semântico diferente do desejado. Uma determinada ação no consumidor pode ser disparada apenas se uma complexa composição de eventos acontecer em diferentes momentos e contextos [2]. Os agentes de processamento de eventos são responsáveis por detectar padrões em eventos, processar esses eventos, transformá-los e validá-los e, finalmente, derivar novos eventos e publicá-los.

2.4.2 Linguagem de Processamento de Eventos

Uma das principais vantagens do CEP é o uso de uma linguagem declarativa para executar o processamento de eventos, uma linguagem de processamento de eventos (EPL - *Event Processing Language*) [55]. Devido à expressividade das EPLs, cenários que anteriormente eram difíceis de implementar podem agora serem especificados com algumas linhas de código, favorecendo a reutilização das soluções [37].

O padrão de especificação da EPL fornece uma linguagem semelhante a SQL com cláusulas SELECT, FROM, WHERE, GROUP BY, HAVING e ORDER BY. Os

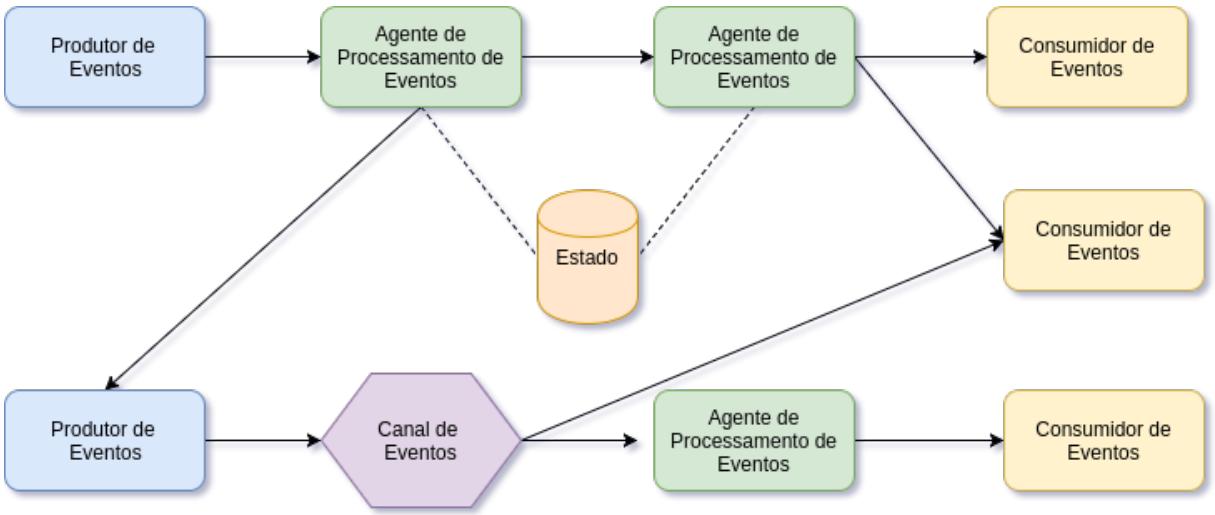

Figura 2.2: Rede de Processamento de Eventos

fluxos de eventos substituem as tabelas como a fonte de dados e os eventos substituem as linhas das tabelas como a unidade básica de dados. Uma vez que os eventos são compostos de dados, os conceitos SQL de correlação através de associações, filtragem e agregação através de agrupamento podem ser implementados.

Para exemplificar o uso da linguagem EPL vamos analisar um exemplo hipotético. Suponhamos que determinado sistema ciente de contexto monitore a temperatura corporal de um paciente através de um sensor que envia dados a cada segundo para a aplicação. A aplicação define 3 tipos de eventos a serem detectados:

- Monitoramento: apenas informa a temperatura a cada 10 segundos.
- Aviso: avisa quando três leituras seguidas estiverem acima de 38º.
- Crítico: alerta quando a temperatura subir abruptamente e o valor inicial for maior que 40°. Se durante cinco leituras consecutivas, a temperatura for aumentando seguidamente e o último valor for 10% maior que o primeiro, considera-se que se atingiu um estado crítico.

Utilizando EPL podemos criar três consultas para modelar cada um dos padrões de eventos acima. A estas consultas serão anexados *listeners* de modo que, quando o padrão for detectado, uma rotina seja ativada. Abaixo apresentamos as declarações EPL para os três tipos de eventos citados.

```
select avg(value) as avg_val
from TemperatureEvent.win:time_batch(10 sec)
```

Listagem 2.1: EPL para evento do tipo monitoramento

```
select * from TemperatureEvent
 match_recognize (
   measures A as temp1, B as temp2, C as temp3
   pattern (A B C)
   define
     A as A.temperature > 38,
     B as B.temperature > 38,
     C as C.temperature > 38)
```

Listagem 2.2: EPL para evento do tipo aviso

```
select * from TemperatureEvent
 match_recognize (
   measures A as temp1, B as temp2, C as temp3, D as temp4,
   E as temp3
   pattern (A B C D E)
   define
     A as A.temperature > 40,
     B as (A.temperature < B.temperature),
     C as (B.temperature < C.temperature),
     D as (C.temperature < D.temperature),
     E as (D.temperature < E.temperature)
      and E.temperature > (A.temperature * 1.10)
```

Listagem 2.3: EPL para evento do tipo crítico

2.5 M-Hub

O M-Hub é um serviço de *middleware*, que executa em dispositivos pessoais móveis, responsável pela descoberta, conexão e aquisição de dados de baixo nível junto a objetos inteligentes (e.g. sensores, atuadores, relógios, robôs, veículos) próximos e acessíveis a partir de tecnologias WPAN (*Wireless Personal Area Network*) de curto alcance, como *Bluetooth Classic* e BLE, ou que estejam embutidos no próprio

dispositivo móvel. Portanto, o M-Hub é um *gateway*, servindo de ponte entre os objetos inteligentes e a Internet.

O M-Hub fornece as seguintes funcionalidades:

1. **Descoberta de sensores próximos:** periodicamente, o M-Hub procura objetos próximos inteligentes que estão anunciando seus IDs e capacidades. Estas informações sobre objetos inteligentes alcançáveis são mantidas no banco de dados do M-Hub.

2. **Conexão com sensores:** dependendo do protocolo WPAN usado, o M-Hub pode precisar primeiramente estabelecer um link de comunicação com o sensor, sobre o qual ele irá emitir solicitações síncronas ou assíncronas para receber periodicamente os dados do sensor.

3. **Protocolo de transcodificação:** Os pacotes de dados recebidos dos sensores podem ter diferentes formatos e codificações. Assim, o M-Hub transcodifica e serializa os dados, antes de transmiti-los. A transcodificação de dados é altamente dependente do tipo, marca e fabricante do sensor.

4. **Caching de dados:** a fim de otimizar a transmissão para a nuvem através da Internet móvel, o M-Hub pode agrupar vários amostras de dados obtidos a partir de vários sensores próximos antes de enviá-los ao gateway em uma única rajada. E para fazer isso, ele armazena em cache as amostras de dados enviadas pelos sensores.

5. **Configuração e Controle dos sensores:**, dependendo do tipo de sensor, o M-Hub pode, eventualmente ou periodicamente, enviar comandos, definições de parâmetros ou requisições de consultas de dados de através da WPAN para os sensores conectados.

6. **Pré-processamento de dados do sensor:** antes de enviar os dados pode ser necessário aplicar uma função de pré-processamento (e.g. transcodificação, formatação, agregação, filtragem, ou comparação com leituras anteriores etc.). Esse pré-processamento é feito no M-Hub.

7. **Carregamento dinâmico de módulos do sensor:** uma vez que não é possível ter módulos internos para todos os sensores que podem estar disponíveis à medida

que o usuário se move, o M-Hub oferece um mecanismo de implantação em tempo de execução e gerenciamento do ciclo de vida de módulos específicos de sensores.

8. **Privacidade** para garantir a privacidade dos dados, o M-Hub implementa um filtro que permite selecionar os dados de contexto que podem ou não ser enviados ao *gateway* SDDL. Dados marcados como "privativos" estarão disponíveis apenas para visualização local no M-Hub View, uma aplicação que permite visualizar os dados recebidos pelo serviço de *middleware* via WPAN.

9. **Processamento nas pontas**: o M-Hub também oferece recursos que permitem que os desenvolvedores de aplicações distribuírem as funcionalidades do seu código entre o dispositivo móvel do usuário a nuvem, movendo parte do processamento das informações de contexto para "as pontas" do sistema. A isso se dá o nome de *In-Network Processing*, uma técnica que ajuda a diminuir a quantidade de informações a ser transmitida para a nuvem e que pode melhorar a escalabilidade do sistema. Para este fim, o M-HUB fornece o suporte para a implantação e gerenciamento de código Java e regras CEP.

10. **Processamento Ciente de Energia**: através de um componente de gerenciamento de energia, o M-Hub monitora o nível da bateria do dispositivo móvel e dispara ações adaptativas que ajustam os comportamento dos seus serviços de acordo com a disponibilidade de energia do dispositivo móvel. Por exemplo, a frequência de publicação dos dados por ser reduzida quando o nível da bateria está baixa (e.g. menor que 20%).

Em relação as WPANs, o M-Hub oferece suporte a conexão com objetos inteligentes via *Classic Bluetooth* (BT 2.0 e 3.0) e *Bluetooth Low Energy* (BLE 4.0). Apesar de seu maior consumo de energia, o *Bluetooth* Clássico ainda é utilizado por uma ampla variedade de dispositivos empregados na área da saúde, tais como o Zephyr BioHarness 3 [1] e Zephyr HxM BT [2]. A BLE está emergindo como uma tecnologia muito promissora. Isso porque ela é mais eficiente em relação co consumo de energia, e também permite uma descoberta mais rápida de dispositivos periféricos. Além disso, a BLE suporta cerca de 2.500 conexões simultâneas. Mas a razão mais importante para

[1] http://zephyranywhere.com/ produtos/BioHarness-3/

[2] http://zephyranywhere.com/products/hxm-bluetooth-heart-rate-monitor/

a escolha da BLE é o fato de ela está disponível em um número crescente de modelos de smartphones (e.g. Android, iOS e BlackBerry) e está sendo incorporada em uma crescente variedade de dispositivos. Por ter uma arquitetura aberta e independente de tecnologia, o M-Hub pode ser estendido para dar suporte a novas WPANs.

A arquitetura do M-Hub, ilustrada na Figura 2.3 é composta por diversos serviços e gerenciadores locais, todos executando em *background* e em paralelo com os aplicativos do usuário.

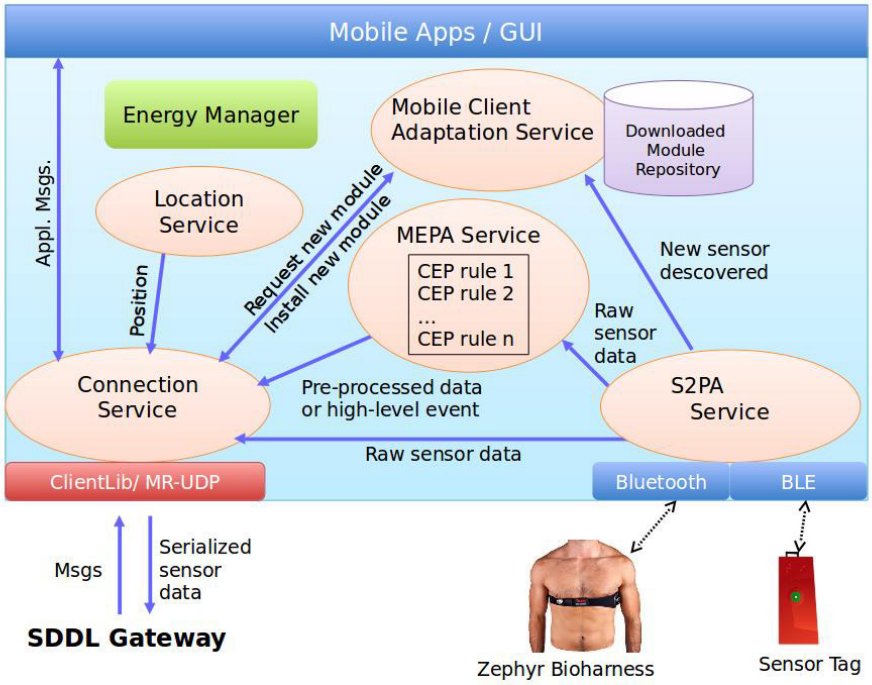

Figura 2.3: Arquitetura do M-Hub

O **LocationService** é responsável por coletar a posição atual do M-Hub. Essa localização é incorporada em todas as amostras de dados publicados pelo M-Hub. Isso permite conhecer a localização de usuários e sensores próximos a ele.

O **S2PA** (*Short-range Sensing, Presence & Actuation*) Service é responsável pela descoberta, monitoramento e registro de sensores e atuadores nas proximidades, fazendo periodicamente varreduras para cada WPAN suportada.

O **ConnectionService** é responsável por manter uma conexão confiável com o gateway SDDL, bem como transmitir dados para a nuvem usando o protocolo MR-UDP.

O **EnergyManager** verifica o nível da bateria do dispositivo periodicamente, e de acordo com a classificação (alta, média e baixa) define as frequências da busca por novos sensores, da consulta ao serviço de localização e do envio dos dados ao gateway SDDL, de modo a minimizar o consumo de energia do dispositivo.

O **Mobile Client Adaptation Service** é responsável pela instanciação e gerenciamento do ciclo de vida do código Java que é baixado dinamicamente. Ele é usado para a implantação de novos módulos de sensores ou funcionalidades da aplicação. Novos módulos são baixados a partir de um serviço em execução na nuvem SDDL. Os módulos baixados são armazenados no dispositivo móvel, a fim de evitar o reenvio de módulos já presentes localmente em caso de uma reinicialização do M-Hub.

O **MEPA (Mobile Event Processing Agent) Service** usa o motor de inferência Esper CEP (Complex Event Processing) para avaliar os fluxos de dados do sensor à procura de certos padrões de correlações de dados como o seu tempo e ordem de ocorrência, expressos como regras. O Esper usa uma linguagem declarativa de processamentos de eventos (EPL) para definir as regras CEP responsáveis por detectar os padrões sobre fluxo de dados. Regras simples podem ter o seguinte formato: `SELECT * FROM SensorDataTicks WHERE SensorName = "HeartRate"`. Essa regra filtra eventos de frequência cardíaca, onde os fluxos de dados de entrada são eventos chamados *SensorDataTicks*, que representam os dados recebidos do objeto inteligente. Já a regra `SELECT * FROM SensorDataTicks (SensorName = "HeartRate") WHERE sensorValue >= 100` filtra apenas os eventos de frequência cardíaca que tenham um valor superior ou igual a 100 batimentos por minuto.

3 Solução Proposta

Uma extensão ao M-Hub tem sido desenvolvida no Laboratório de Sistemas Distribuídos Inteligentes da Universidade Federal do Maranhão ((LSDi-UFMA) denominada Camada de Distribuição de Dados de Contexto (CDDL - *Context Data Distribution Layer*) que adiciona ao *middleware* suporte à Qualidade de Contexto e mecanismos de registro e descoberta de serviços de contexto com especificação de requisitos de QoC, bem como provisionamento e monitoramento das qualidades da informação e do serviço de distribuição.

O M-Hub e o CDDL são executados usualmente em uma mesma máquina mas podem também serem executados separadamente. Isto significa que o M-Hub pode publicar dados de contexto utilizando-se de outros serviços de distribuição e, por sua vez, o CDDL pode publicar dados coletados por outro sistema de aquisição de dados de contexto. Enquanto o M-Hub executa em dispositivos pessoais móveis (e.g *smartphones* e *tablets*), existem atualmente duas versões do CDDL: uma para dispositivos Android e outra para computadores que possuam uma Máquina Virtual Java (*Java Virtual Machine - JVM*) instalada.

O CDDL é um *middleware publicador/subscritor* que fornece às aplicações clientes a capacidade de atuar como produtoras e consumidoras de dados de contexto. O *middleware* dá suporte a políticas de qualidade do serviço de distribuição dos dados de contexto que permitem às aplicações especificarem diversos parâmetros que expressam seus requisitos de QoC, tanto para o envio como para o recebimento das informações de contexto. Além disso, ele adiciona metadados de qualidade de contexto às informações publicadas. Outra funcionalidade fornecida pelo CDDL é a descoberta de serviços através do uso de diretórios de serviços. A aplicação pode consultar estes diretórios especificando requisitos de QoC em seus critérios de busca. Um mecanismo de filtragem dos dados é fornecido com o qual as aplicações podem especificar filtros para os dados publicados e/ou recebidos, de modo a diminuir a quantidade de dados processados pela aplicação. Finalmente, o *middleware* fornece componentes para monitoramento da QoC durante a execução das aplicações, possibilitando que as aplicações consumidoras das informações de contexto possam

receber diversos tipos de notificações sobre a variação da QoC como, por exemplo, serem notificadas quando seus requisitos de QoC não puderem ser mais cumpridos pelo provedor de serviço.

Como vimos, o CDDL é um *middleware* com diversas funcionalidades que, em conjunto, permitem o gerenciamento da distribuição dos dados de contexto e subscrição e publicação destes dados pelas aplicações consumidoras. Este trabalho de mestrado se concentra na implementação das funcionalidades de avaliação da QoC, filtragem e monitoramento.

3.1 Requisitos do CDDL

A fim de fornecer um amplo suporte ao desenvolvimento de aplicações da IoMT com requisitos de QoC, sem deixar de abranger também o desenvolvimento de aplicações da IoT que não necessariamente tenham requisitos de QoC e/ou de mobilidade irrestrita, foi definido que a CDDL deveria atender, além das funcionalidades já incorporadas ao M-Hub, os seguintes requisitos:

- **Suporte à Distribuição de Dados Local e Remota**: a solução deve prover mecanismos de distribuição de dados, a fim de possibilitar o compartilhamento de informações entre produtores e consumidores de contexto. O *middleware* deve dar suporte tanto a distribuição local (produtores e consumidores executando no mesmo dispositivo) como a distribuição remota (produtores e consumidores executam em dispositivos distintos).

- **Suporte ao Registro e Descoberta Distribuída de Serviços**: a solução deve prover mecanismos que permitam descrever as características dos serviços de contexto disponíveis, a fim de que possam ser registrados em diretórios de serviços locais (registram serviços disponíveis no mesmo dispositivo) ou globais (executado em uma infraestrutura de nuvem e que mantém um registro de todos os serviços descobertos que são fornecidos por um grupo de instâncias CDDL - um domínio CDDL). As aplicações cientes de contexto devem descobrir os serviços disponíveis por meio da realização de consultas a esses diretórios.

- **Modelo de Programação Uniforme e Independente de Localização**: esse requisito expressa que, independentemente da localização dos produtores, consumidores, diretórios de serviços e *brokers*, sejam eles locais ou remotos, as interfaces de programação utilizadas para registro, publicação, consulta e subscrição de dados de contexto devem ser uniformes, ou seja, a maneira de programar uma aplicação que consome dados de contexto de origem local deve ser a mesma utilizada para desenvolver uma aplicação que consome dados de contexto de origem remota.

- **Suporte à Entrega Confiável de Dados em Cenários de Mobilidade**: a fim de abranger também a distribuição de informações de contexto em cenários de mobilidade irrestrita, a solução deve prover mecanismos que minimizem a perda de dados cuja entrega não foi confirmada, eventualmente devido a problemas relacionados à conectividade fraca ou intermitente dos *gateways* locais e/ou aplicações consumidoras com a Internet.

- **Provisionamento e Monitoramento de Qualidade da Informação**: a solução deve prover mecanismos que permitam inserir na informação de contexto o valor de determinados atributos que descrevem a qualidade das mesmas (metadados de QoI) e/ou meios para que o *middleware* possa calculá-los dinamicamente. Os mecanismos de registro e descoberta de serviços devem levar em consideração a QoI. Portanto, os diretórios de serviços devem estar cientes da qualidade das informações disponíveis. As consultas de contexto devem permitir também a especificação de requisitos de QoC dos consumidores. Além disso, devem ser providos mecanismos de monitoramento da qualidade da informação, a fim de permitir que as aplicações consumidoras sejam notificadas quando variações significativas na QoI ocorrerem.

- **Provisionamento e Monitoramento de Qualidade do Serviço de Distribuição**: a solução deve prover mecanismos que permitam que produtores e consumidores configurem políticas que definem a qualidade do serviço de distribuição dos dados de contexto, possibilitando que eles controlem como e quando as informações de contexto serão publicadas, recebidas e armazenadas. Os mecanismos de registro e descoberta de serviços devem levar em consideração também a necessidade de armazenar e consultar informações sobre a qualidade

do serviço de distribuição. Para algumas políticas de QoS, devem ser providos mecanismos de monitoramento, a fim de que publicadores e subscritores sejam notificados quando a qualidade exigida deixar de ser atendida.

- **Filtragem das informações**: a solução deve prover mecanismos que possibilitem a filtragem das informações com base no conteúdo de seus atributos, inclusive os meta-dados de QoI. A filtragem deve permitir ao publicador especificar quais dados podem ser publicados, e ao subscritor definir quais dados recebidos podem ser repassados para a aplicação consumidora.

O M-Hub utiliza uma *engine* CEP Asper [30] (uma versão modificada da *engine* CEP Esper [1] para Android para o processamento do fluxo de informações de contexto gerenciados pelo *middleware*. Os componentes *Filter*, *Monitor*, *QoCEvaluator* e *Global Direcotry Service* instanciam agentes de processamento de eventos (EPA) para executar suas funcionalidades. O componente *Local Directory Service* utiliza o EPA instanciado pelo *QoCEvaluator*.

Este trabalho de mestrado se concentra na implementação dos três últimos requisitos. Os componentes que implementam tais requisitos estão destacados na figura da arquitetura na cor cinza, sendo que, para algumas políticas de QoS, os mecanismos de monitoramento e notificação também são implementados nos componentes que definem a política.

A seção seguinte apresenta a arquitetura do *middleware*. Nela é possível identificar os componentes que implementam as funcionalidades necessárias para atender os requisitos levantados.

3.2 Arquitetura e Funcionalidades dos Componentes

A Figura 3.1 ilustra a arquitetura estendida do M-Hub/CDDL e seus novos componentes. Os novos componentes e suas funcionalidades dentro da arquitetura são explicados em seguida.

Os componentes da camada M-Hub já foram discutidos na Seção 2.5. Passamos a detalhar os componentes da infraestrutura pertencentes à CDDL:

[1] url=http://www.espertech.com/esper/

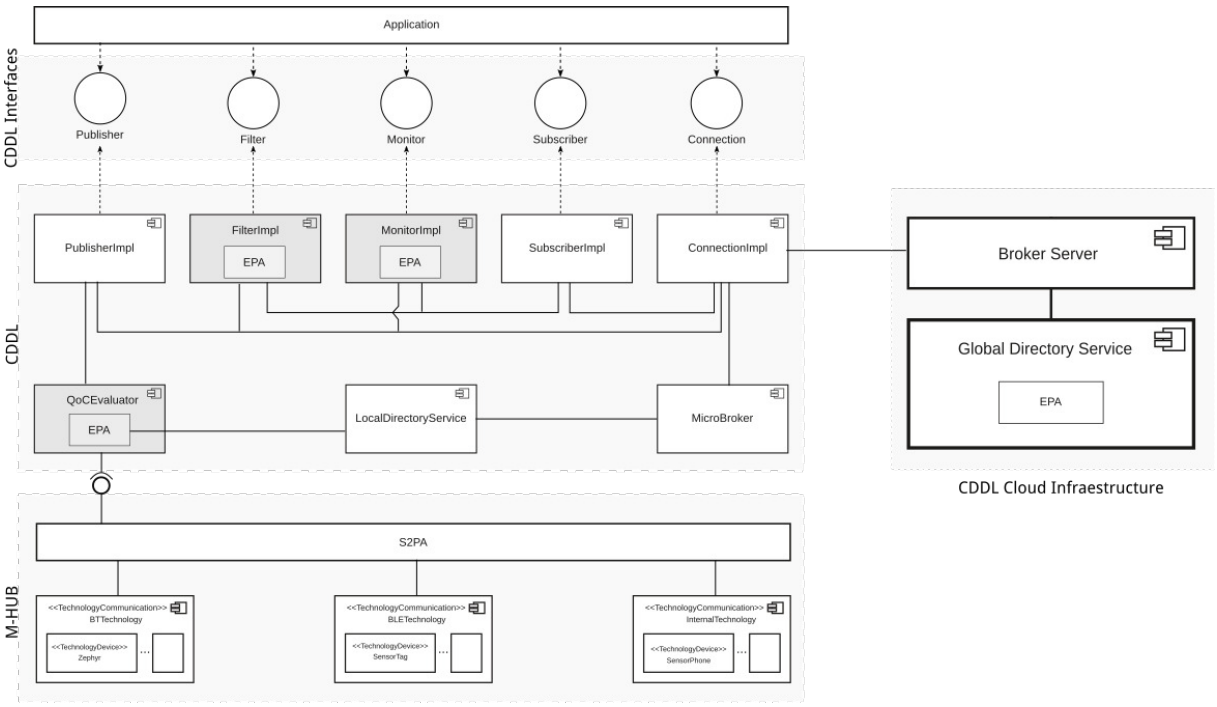

Figura 3.1: Arquitetura estendida do M-Hub/CDDL

- **Publisher**: componente responsável pela publicação de dados, que podem ser oriundos do S2PA ou fornecidos pela camada de aplicação. Do ponto de vista das aplicações consumidoras, cada *Publisher* instanciado é provedor de serviços de contexto. Além da publicação de dados de contexto enriquecidos com meta-dados de QoI, esse componente também pode ser utilizado para publicar consultas e respostas de contexto. A entrega dos dados ao *broker* é realizada de acordo com um conjunto de parâmetros e políticas de QoS implementadas pelo componente.

- **Subscriber**: componente responsável pela subscrição em tópicos. Além do recebimento de dados de contexto enriquecidos com meta-dados de QoI, o *Subscriber* também pode ser usado para receber consultas e respostas de contexto. A entrega dos dados a camada de aplicação é feita de acordo com alguns parâmetros e políticas de QoS implementadas pelo componente.

- **Local Directory Service**: componente responsável pelo gerenciamento dos registros dos provedores de serviços de contexto que executam no próprio dispositivo. Esses registros incluem dados sobre os tipos de contexto disponíveis, a qualidade média das informações publicadas e o nível da qualidade do serviço de distribuição oferecido pelo provedor.

- **Connection**: componente responsável pelo gerenciamento de conexões e sessões estabelecidas pelos clientes (publicadores e subscritores) com os *brokers*, sejam locais ou remotos.

- **Micro Broker**: versão modificada de um *Broker Server* voltado para dispositivos móveis da plataforma *Android*. Sua função é intermediar a comunicação entre publicadores e subscritores com base na definição de sequências e filtros de tópicos. Por padrão, esse componente é configurado para aceitar somente conexões locais. Desse modo, a troca de mensagens é possível apenas entre aplicações que executam no mesmo dispositivo. Entretanto, o *Micro Broker* também pode ser configurado para aceitar conexões externas, possibilitando assim a comunicação entre aplicações que executam em dispositivos diferentes, desde que conectados na mesma WLAN.

- **QoC Evaluator**: componente que recebe dados de contexto enriquecidos com meta-dados de QoI vindos do S2PA. Uma das responsabilidades do *QoC Evaluator* é computar dinamicamente o valor de alguns parâmetros de QoI, que não são fornecidos na etapa de aquisição. Sua outra função é calcular periodicamente a qualidade média dos dados de contexto fornecidos por cada sensor. Essa média é uma informação relevante para algumas aplicações consumidoras, que podem adotá-la com um critério de seleção dos serviços de contexto disponíveis. Essa informação é armazenada nos diretórios de serviços. Após a avaliação das QoIs, estas são adicionadas na forma de metadados às informações de contextos que são encaminhadas ao componente *Publisher*.

- **Filter**: componente que filtra as informações com base no conteúdo de seus atributos, inclusive os meta-dados de QoI. Esse componente pode ser usado tanto pelo *Publisher*, para definir quais dados podem ser publicados, como pelo *Subscriber*, para definir quais dos dados recebidos podem ser repassados para a aplicação consumidora. A filtragem por conteúdo não substitui a filtragem por tópicos executada pelos *brokers*, sendo apenas uma segunda camada de filtragem.

- **Monitor**: componente responsável pelo monitoramento de eventos de interesse da aplicação. Esse monitoramento é baseado em regras. Para cada regra de monitoramento é associado um *listener* pelo qual a aplicação é notificada em caso de ocorrência do evento. Este componente pode ser utilizado

para o monitoramento de QoI e QoS. Entretanto, algumas políticas de QoS, especificamente, são automonitoradas, ou seja, a própria política possui um mecanismo de monitoramento padrão que notifica a aplicação caso a qualidade do serviço seja violada.

- **Server Broker**: componente cuja função é intermediar a comunicação entre publicadores e subscritores conectados à nuvem. Esse componente possui maior capacidade de distribuição de dados que o *Micro Broker*, pois executa num hardware com recursos menos limitados.

- **Global Directory Service**: componente responsável pelo registro dos provedores de serviços conectados a nuvem que formam um domínio CDDL. A base de dados do *Global Directory Service* é atualizada a partir das listas de serviços disponíveis que são publicadas periodicamente pelo diretório local que é executado em cada máquina do domínio CDDL.

Este trabalho de mestrado se concentra no desenvolvimento dos componentes *QoC Evaluator*, *Filter* e *Monitor*. Eles são responsáveis pela avaliação da QoC, filtragem baseada em conteúdo e monitoramento de eventos, respectivamente.

3.3 Interfaces e Abstrações de Programação

Para utilizar o *middleware*, o desenvolvedor conta com uma API de programação que expõe todas as funcionalidades disponíveis para que ele possa publicar informações de contexto, subscrever os serviços dos quais deseja receber informações, estabelecer filtros para publicar ou receber somente determinadas informações de contexto, consultar os serviços de diretórios para descobrir quais informações de contexto estão disponíveis para subscrição, monitorar o fluxo de informações de contexto que estão sendo publicadas ou subscritas e estabelecer condições que, uma vez atendidas, gerem notificações. O desenvolvedor pode utilizar, em seus critérios de consultas, filtragens ou monitoramento nao somente elementos relativos aos dados e serviços de contexto, mas tambem à metainformaçao de QoC associada aos mesmos, como sera visto em exemplos ao longo deste texto. Isto possibilita um amplo controle sobre como o *middleware* envia e recebe os dados de

contexto. A seguir apresentamos as interfaces e abstrações de programação dos principais componentes do *middleware*.

3.3.1 Consulta aos Serviços Disponíveis por Smart Objects

As aplicações que utilizam o *middleware* podem realizar consultas para descobrir quais serviços estão disponíveis para subscrição. A consulta é realizada através do componente *Publisher*. Os serviços de diretórios locais e globais são responsáveis por responder às consultas. O método void `query(QueryType queryType, String epl)` do componente *Publisher* é utilizado para consultar os serviços de diretórios de modo a obter uma lista dos serviços disponíveis. Os parâmetros são o tipo de consulta e a cláusula de consulta (uma string contendo uma regra expressa na linguagem EPL da *engine* CEP Asper) que estabelece quais os critérios da consulta).

Existem dois tipos de consulta: instantâneas e contínuas. As consultas instantâneas retornam os publicadores e serviços disponíveis que satisfazem os critérios de busca no momento em que a consulta é realizada. Após a resposta ser entregue, a consulta é encerrada. As consultas contínuas, por sua vez, além de retornar quais publicadores e serviços estão disponíveis no momento da consulta, continuam em execução e periodicamente entregam à aplicação que realizou a consulta os resultados de novos publicadores e serviços que por ventura sejam descobertos e atendam aos critérios de busca. Ambas as consultas podem ser canceladas a qualquer momento pela aplicação. Além das consultas, uma outra maneira de descobrir novos serviços é assinar tópicos especiais onde os serviços de diretórios locais e globais publicam periodicamente uma lista atualizada dos serviços registrados.

A cláusula de consulta (segundo parâmetro do método `query()`) define os critérios de consulta. Os critérios podem ser estabelecidos com base nas informações de contexto em si ou nos parâmetros de QoC. Por exemplo, caso a aplicação deseje saber quais os serviços de localização estão disponíveis (de qualquer publicador) e cuja idade das informações seja no máximo de 1 segundo, ele pode utilizar a declaração EPL apresentada na listagem 3.1:

```
1 publisher.query(QueryType.INSTANTANEOUS, ``select * from
     ServiceInformationMessage where serviceName = 'LOCATION' and age < 1000)
```

Listagem 3.1: Exemplo de critério de consulta

`ServiceInformationMessage` é uma estrutura de dados utilizada pelos serviços de diretórios para armazenar informações sobre os serviços publicados. Como neste exemplo o tipo de consulta é instantâneo, a resposta compreenderia apenas os serviços disponíveis no momento da publicação da consulta.

3.3.2 Consumindo Dados de Contexto

Para subscrever informações de contexto, uma aplicação deve obter uma referência para uma instância do componente *Subscriber*. Para tal, ela deve utilizar a interface `Subscriber` apresentada na Figura 3.2 juntamente com a interface `Client` da qual `Subscriber` herda diversos métodos. A seguir descrevemos a utilização dos principais métodos da interface `Subscriber`:

(a) O método `void subscribeServiceByPublisherId(String publisherId)` é utilizado para a aplicação subscrever as informações de contexto que estão sendo publicadas por um provedor específico. Ele recebe como parâmetro, o ID do publicador. A aplicação pode conhecer de antemão o ID do publicador ou receber este identificador a partir de uma consulta aos diretórios de serviços. Todas as informações de contexto que este publicador estiver disponibilizando são assinadas. Na listagem 3.2 podemos ver um exemplo de subscrição dos serviços de um publicador específico.

```
1 ...
2 // informa o listener que receberá a informação de contexto subscrita.
3 subscriberListener = new ExampleSubscriberListener();
4 subscriber.setSubscriberListener(subscriberListener);
5
6  // subscreve os serviços do publicador identificado pelo id '
      johndoe@example.com'
7 subscriber.subscribeServiceByPublisherId("johndoe@example.com");
8 ...
```

Listagem 3.2: Exemplo de subscrição de informação de contexto

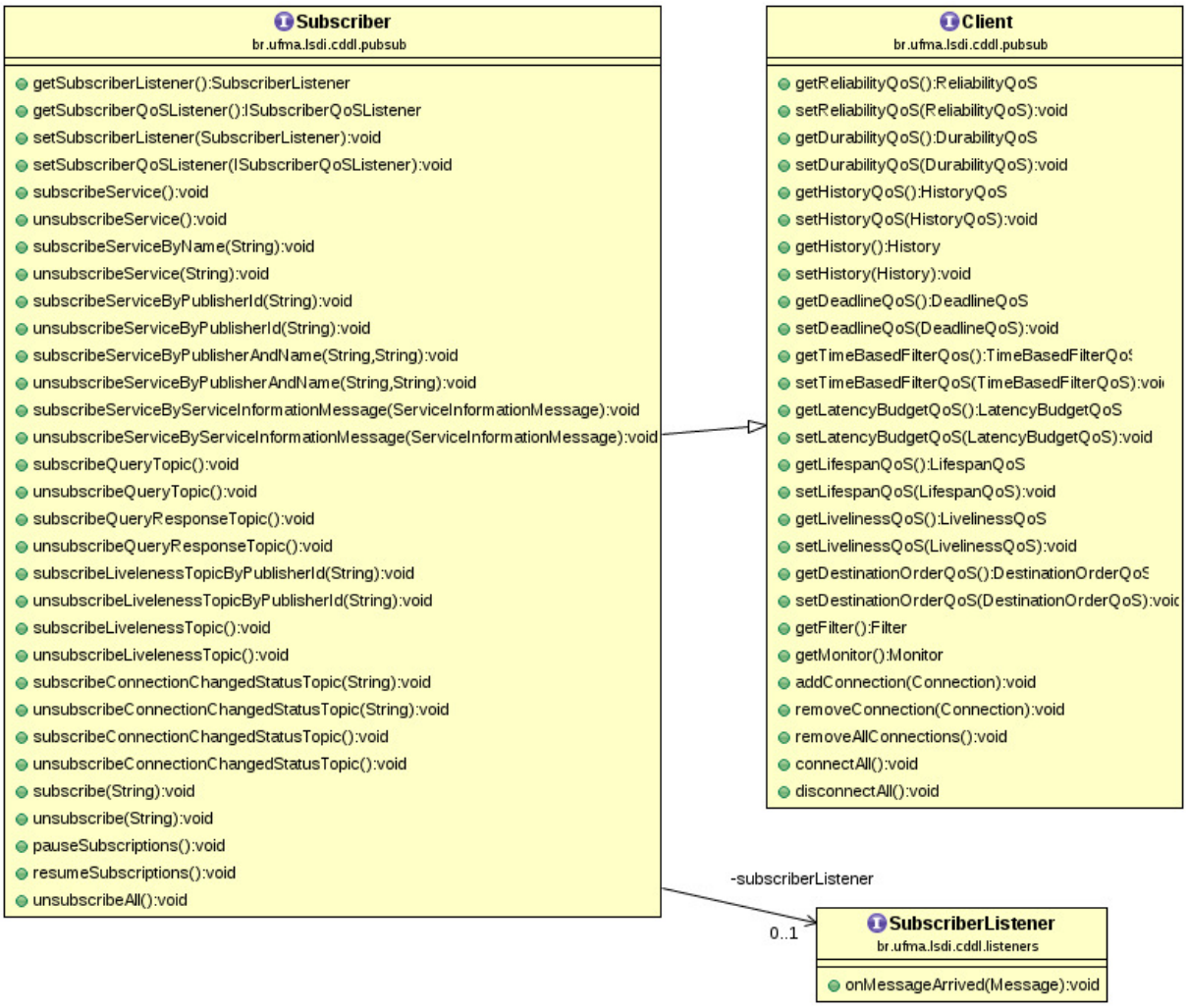

Figura 3.2: Interface Subscriber

(b) O método `vpid subscribeServiceByName(String serviceName)` é utilizado para a aplicação subscrever um serviço específico. Neste caso, a aplicação recebe as informações de contexto de todos os publicadores que estão publicando este serviço. Por exemplo, caso a aplicação execute a seguinte instrução `subscriber.subscribeServiceByName('TEMPERATURE')`, ela receberá o serviço especificado de todos os publicadores disponíveis.

(c) Se a aplicação deseja subscrever um serviço especifico de um publicador determinado, ela deve utilizar o método `void subscribeServiceByPublisherIdAndName(String publisherId, String serviceName)`. Nesse caso, ela informa o ID do publicador e o nome do serviço.

Conforme pode-se observar na Figura 3.2, a interface disponibiliza diversos métodos do tipo `void subscribe*()`. Para cada método `void`

`subscribe*()`, o `Subscriber` possui um método `void unsubscribe*()` correspondente de modo a remover a subscrição referente. Por exemplo, o método `void unsubscribeServiceByName(String serviceName)` cancela a subscrição de um serviço pelo nome.

(d) Após a aplicação se subscrever para receber informações de contexto, o *middleware* deve ter uma maneira de enviar essas informações para o subscritor. A aplicação, portanto, deve informar um *listener* que implementa a interface `SubscriberListener` para o *middleware* utilizar como receptor das informações de contexto publicadas. A aplicação deve utilizar o método `void setSubscriberListener(SubscriberListener listener)` para fornecer uma referência ao *listener*. Quando uma informação de contexto do tipo assinado pelo subscritor é recebida pelo *middleware*, este chama o método `void onMessageArrived(Message message)` do *listener* passando a mensagem publicada que poderá então ser utilizada pela aplicação.

O *listener* especificado com o método `setSubscriberListener()` é utilizado também pelo *middleware* para responder as consultas publicadas pela aplicação. Quando uma consulta é feita aos serviços de diretórios, estes armazenam a resposta numa estrutura de dados chamada `QueryResponseMessage` que é enviada para o *listener*. Uma `QueryResponseMessage` contém uma lista de objetos do tipo `ServiceInformationMessage` que, por sua vez, contêm as informações referentes a cada serviço (nome do publicador, nome do serviço e dados de QoC). A listagem 3.3 mostra um exemplo de *listener* que recebe tanto informações de contexto como respostas a consultas efeituadas.

```
1  ...
2  @Override
3  public void onMessageArrived(Message message) {
4
5      // As informações de contexto ou as respostas de consultas
6      // são recebidas neste método.
7
8      if(message instanceof SensorDataMessage) {
9
10         final SensorDataMessage msg =
11           (SensorDataMessage) message;
12
```

```
13          Log.d(TAG, "Nova informação de contexto:" + msg);
14          Log.d(TAG, "Serviço: " + msg.getServiceName());
15          Log.d(TAG, "Valor: " + msg.getServiceValue());
16          Log.d(TAG, "Acurácia: " + msg.getAccuracy());
17
18      }
19
20      else if(message instanceof  QueryResponseMessage) {
21
22          final QueryResponseMessage
23            response = (QueryResponseMessage) message;
24
25          Log.d(TAG, "Resposta da consulta:" + response);
26
27          // testa o código de retorno da consulta
28          if (response.getQueryMessage()
29            .getReturnCode() == returnCode) {
30
31              // No exemplo abaixo, a aplicação esta subscrevendo
32              // os serviços retornados pela consulta
33              for (ServiceInformationMessage sim :
34                 queryResponseMessage
35                   .getServiceInformationMessageList()) {
36
37                     subscriber.subscribeSensorDataTopic
38                       ByServiceInformationMessage(sim);
39              }
40
41          }
42
43      }
44
45  }
46  ---
```

Listagem 3.3: Exemplo de listener para receber informações de contexto

(e) Um mesmo subscritor pode se conectar a vários *brokers*, incluindo o *micro broker* local. Dessa forma, a aplicação pode receber dados de contexto de mais de um *broker* com apenas uma instrução *subscribe*. O método `void`

`addConnection(Connection connection)` é utilizado para conectar o subscritor com um *broker*.

(f) Para desconectar um subscritor de um *broker* a aplicação deve utilizar o método `void removeConnection(Connection connection)`. A partir deste instante, o subscritor para de receber informações deste *broker* em particular.

A interface `Subscriber` estende a interface `Client` que disponibiliza métodos que possibilitam ao subscritor informar diversos parâmetros de QoS que gerenciam como os dados de contexto são subscritos. A aplicação pode ajustar ou ler os parâmetros de confiabilidade (*reliabilityQoS*), durabilidade (*durabilityQoS*), histórico (*historyQoS*), tempo de vida (*deadlineQoS*), ordem de destino (*destinationOrderQoS*), dentre outros.

3.3.3 Publicando Dados de contexto

Para publicar informações de contexto, uma aplicação deve obter uma referência para uma instância do componente *Publisher* utilizando a interface `Publisher`. Assim como a interface `Subscriber`, `Publisher` estende a interface `Client`.

(a) O método `void publish(Message message)` é utilizado para publicar uma informação de contexto. Ele recebe como parâmetro, a mensagem que a aplicação deseja publicar. A informação propriamente dita é criada utilizando-se a classe `Message`, com a qual podemos informar o nome do serviço, seus valores e informações de QoC, quem está publicando a informação, dentre outras informações. Quando a aplicação chama o método *publish*, ela realiza uma chamada assíncrona e o *middleware* realiza todos os procedimentos necessários para enviar os dados para os *brokers* nos quais o publicador está conectado.

(b) Um mesmo publicador pode se conectar a vários *brokers*, incluindo o *micro broker* local. Dessa forma, a aplicação pode publicar dados de contexto para mais de um *broker* com apenas uma instrução *publish*. O método `void addConnection(Connection connection)` é utilizado para conectar o publicador com um *broker*.

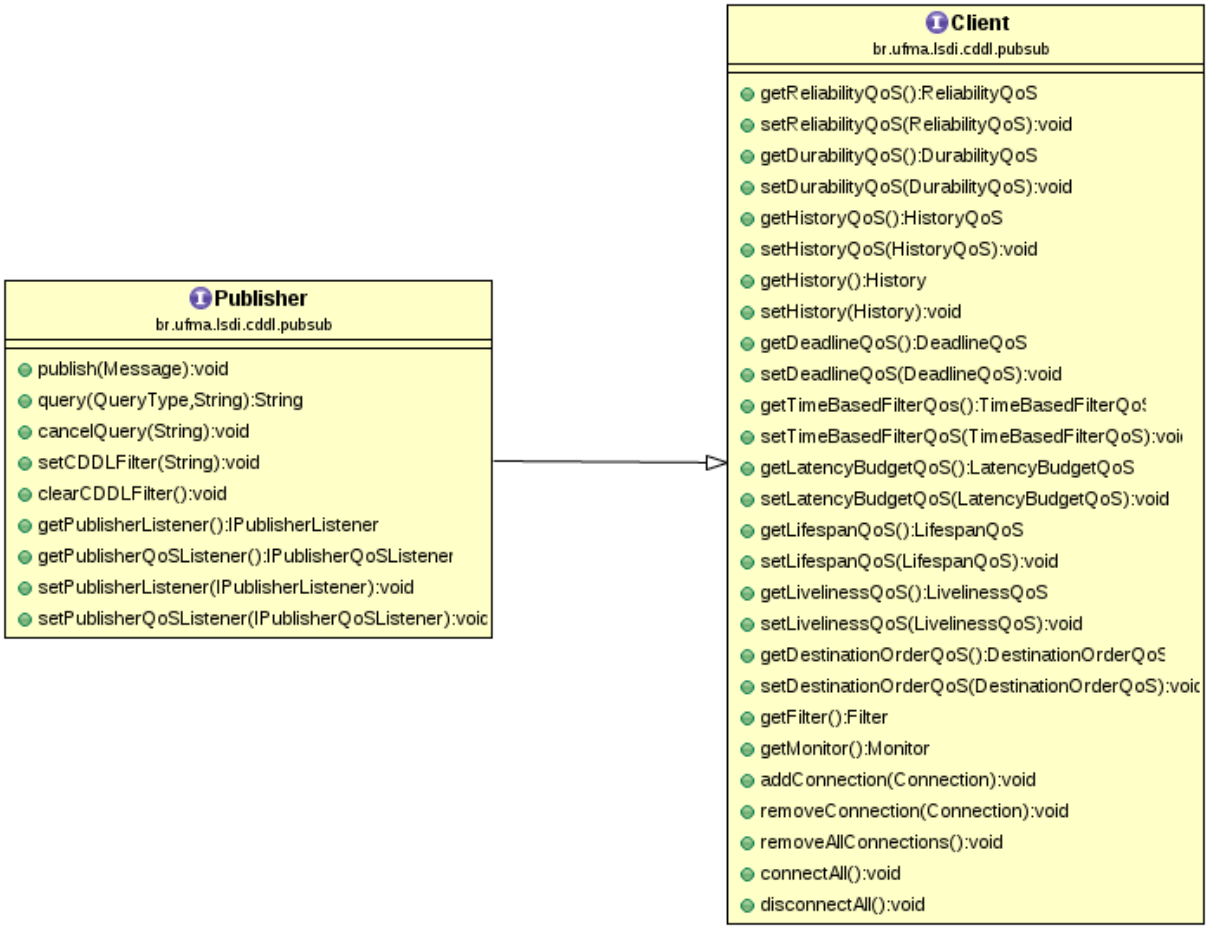

Figura 3.3: Interface Publisher

O CDDL é muito flexível em relação às configurações de como os fluxos de dados de contexto podem ser estabelecidos. Um *Publisher* ou *Subscriber* pode se conectar a um *Micro Broker* rodando na mesma máquina em que estão sendo executados ou pode estabelecer uma comunicação com um *Micro Broker* sendo executado no CDDL em outra máquina. Eles também podem ser conectados a um *Server Broker* rodando na nuvem CDDL. Os *Publishers* e *Subscribers* podem também estabelecerem múltiplas conexões. Se estiverem conectados apenas a *Micro Brokers*, as consultas realizadas terão **escopo local** enquanto que, caso estejam conectados a *Server Brokers*, as consultas terão **escopo global**.

(c) Para desconectar um publicador de um *broker* a aplicação deve utilizar o método `void removeConnection(Connection connection)`. A partir deste instante, o publicador para de enviar informações para este *broker* e os serviços de diretórios que utilizam esta conexão removem este publicador de sua base de dados.

Além dos métodos acima, a interface `Client` disponibiliza métodos que possibilitam ao publicador informar diversos parâmetros de QoS que gerenciam como os dados de contexto são publicados. A aplicação pode ajustar ou ler os parâmetros de confiabilidade (*reliabilityQoS*), durabilidade (*durabilityQoS*), histórico (*historyQoS*), tempo de vida (*deadlineQoS*), ordem de destino (*destinationOrderQoS*), dentre outros.

3.3.4 Uso de filtros

O componente *Filter* é utilizado pela aplicação para especificar regras que determinam quais dados de contexto e/ou anúncios de serviços podem de fato ser publicados pelo *Publisher*, bem como para definir quais dados e/ou notificações de descoberta de serviços recebidos pelo *Subscriber* devem ser realmente encaminhados para a aplicação consumidora. No publicador, o filtro tem a finalidade de diminuir o volume de dados a ser transmitido. Isso é útil, por exemplo, quando o *gateway* está utilizando redes móveis, onde o volume de tráfego é limitado pelo tamanho do pacote de dados do assinante. Outra razão para filtrar os dados no ato de publicá-los é preservar a privacidade do usuário. Por exemplo, o filtro poderá impedir a publicação da localização em determinadas coordenadas em que o usuário do *smartphone* não deseja ser rastreado. No subscritor, o filtro ajuda a reduzir o volume de dados a serem repassados para a camada de aplicação. Por exemplo, uma aplicação pode assinar dados de localização mas desejar receber apenas aqueles com acurácia menor ou igual a 5 metros (Figura 3.4). Outro exemplo, se a aplicação consome dados de localização e deseja ser notificada apenas quando as coordenadas correspondem a um lugar específico, então ela pode especificar um filtro que não deixe passar dados de localização irrelevantes.

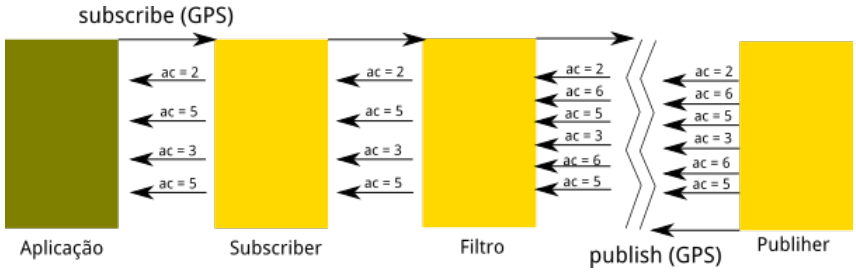

Figura 3.4: Exemplo de filtragem

O componente *Filter* desempenha um papel importante no provisionamento de QoC pois permite que os dados sejam filtrados considerando os metadados

de qualidade associados. O *Filter* utiliza um *engine* CEP Asper para processar as informações de contexto. O mecanismo de filtragem é especificado por meio de regras escritas na linguagem EPL da *engine* CEP Asper. Como exemplo, considere a declaração EPL apresentada na listagem 3.4 que filtra um fluxo de dados de localização exigindo uma acurácia menor ou igual a 1.

```
1 SELECT * FROM SensorDataMessage WHERE serviceName = "Localization" AND
    accuracy <= 1
```

Listagem 3.4: Exemplo de EPL de filtragem

A interface `Filter` utilizada pela aplicação para acessar as funcionalidades do componente *Filter* é mostrada na Figura 3.5. Para obter uma referência à interface Filter, a aplicação utiliza o método `Filter getFilter()` das interfaces `Publisher` e `Subscriber`. A seguir descrevemos a utilização dos métodos da interface `Filter`:

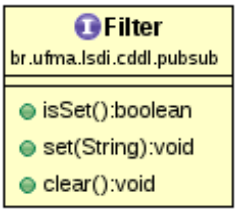

Figura 3.5: Interface Filter

(a) O método `void set(String epl)` é utilizado para especificar uma cláusula de filtragem (uma string contendo uma regra expressa na linguagem EPL da *engine* CEP Asper) que será aplicada a todas as informações que passam pelo filtro.

(b) O método `void clear()` remove o filtro corrente. Apenas um filtro pode ser especificado por vez para cada publicador/subscritor.

(c) O método `boolean isSet()` permite à aplicação verificar se existe um filtro aplicado em um determinado momento.

3.3.5 Monitorando serviços e dados de contexto

O componente *Monitor* permite a especificação de regras CEP que ativem notificações de eventos a partir de critérios estabelecidos pela aplicação.

A especificação das regras também é feita em EPL. Uma das aplicações do monitoramento é a detecção de eventos relacionados às variações de QoC. Por exemplo, se uma aplicação escolheu usar um GPS devido à sua alta precisão e depois de um tempo a acurácia é reduzida, então a aplicação precisa ser notificada para que possa selecionar outro provedor de serviço disponível, uma vez que a qualidade do provedor em uso pode não atender mais os seus requisitos.

O *Monitor* utiliza um *engine* CEP Asper para monitorar as informações de contexto. A aplicação cliente solicita ao *Subscriber* a ativação do monitoramento na forma de uma regra expressa na linguagem EPL que especifica as condições para os parâmetros de QoC que deseja monitorar. O *Publisher* por sua vez, ativa o monitoramento passando ao *Monitor* a solicitação da aplicação cliente e um *listener* para onde o *Monitor* irá enviar os eventos sobre alterações na QoC de acordo com a cláusula informada. Quando um monitoramento está ativo, todas as amostras de informações de contexto que chegam ao *Subscriber* são avaliadas pela regra e, caso a regra seja satisfeita, o *Monitor* envia uma notificação para a aplicação através do *listener* informado.

A interface `Monitor` utilizada pela aplicação para acessar as funcionalidades do componente *Monitor* é mostrada na Figura 3.6. Para obter uma referência para a interface `Monitor`, a aplicação utiliza o método `Monitor getMonitor()` das interfaces `Publisher` e `Subscriber`. A seguir descrevemos a utilização dos seus métodos:

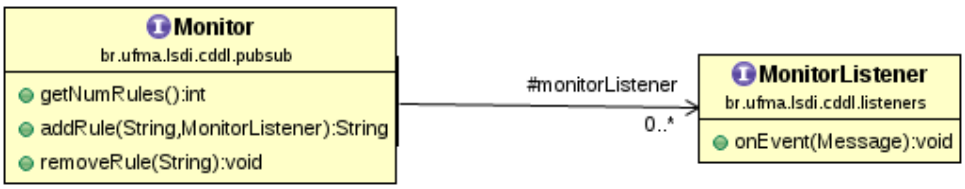

Figura 3.6: Interface Monitor

(a) O método `String addRule(String epl, MonitorListener listener)` é utilizado para adicionar uma string contendo uma regra de monitoramento expressa na linguagem EPL da *engine* CEP Asper. O segundo parâmetro deste método é um *listener* que implementa a interface `MonitorListener`. Este *listener* é usado pelo monitor para enviar as notificações quando a regras de monitoramento são satisfeitas. O *listener*

deve implementar o método `void onEvent(Message message)` que será chamado pelo monitor passando a mensagem que satisfez a regra de monitoramento.

(b) O método `void removeRule(String ruleId)` remove uma regra de monitoramento. Desse modo, a aplicação deixa de receber notificações para a regra em questão.

(c) O método `int getNumRules()` informa quantas regras de monitoramento estão ativas em determinado momento.

3.4 Aspectos de Implementação dos Componentes QoCEvaluator, Monitor e Filter

Como descrito na Seção 1.2.1, o escopo desse trabalho de mestrado é o desenvolvimento de mecanismos de provisionamento de QoC a serem integrados ao *middleware* M-Hub relativos às seguintes funcionalidades: avaliação de parâmetros de QoC, monitoramento de QoC e filtragem de informações baseadas em critérios de QoC. Por isso, nesta seção descrevemos os componentes que implementam estas funcionalidades: o *QoCEvaluator*, o *Monitor* e o *Filter*, respectivamente.

3.4.1 QoC Evaluator

Após os dados dos sensores serem coletados pelo M-Hub através dos *technology devices* (e.g. SensorPhone, Zephyr, SensorTag) (ver Figura 3.1), eles são enviados para o componente *QoC Evaluator*. Esse componente é responsável por calcular e injetar na informação de contexto os valores dos parâmetros de QoC que não são fornecidos prontamente pelos sensores. Os parâmetros de QoC inseridos na informação de contexto são: acurácia, tempo de medição, atributos disponíveis, localização da fonte, intervalo de medição, resolução numérica e idade. Estes parâmetros são calculados como segue:

- **Acurácia**: alguns sensores informam a acurácia da informação de contexto que enviam ao *middleware*. Nesse caso, o *QoC Evaluator* a armazena seu valor no

atributo *accuracy*. No caso do sensor não disponibilizar esta informação, o *QoC Evaluator* deixa esse atributo com valor nulo.

- **Tempo de medição**: O tempo de medição também é um dado que alguns sensores disponibilizam e que, neste caso, o *QoC Evaluator* grava no atributo *measurement time*. Caso o tempo de medição não seja fornecido pelo sensor, o *QoC Evaluator* calcula o atributo como sendo o momento em que a informação chegou ao *gateway*.

- **Atributos disponíveis**: O *technology device* que lê os dados de determinado tipo de sensor, pode informar o número de atributos que determinada informação de contexto deve possuir. Por exemplo, o componente *SensorPhone* do *M-Hub* é o responsável pela leitura dos dados dos sensores internos do dispositivo onde o *middleware* está instalado. No caso do sensor de localização, o *technology device* informa que são armazenados três valores (longitude, latitude e altitude). Ao receber os dados deste componente, através do *S2PA*, o *QoC Evaluator* informa quantos atributos foram realmente recebidos (no caso da localização, os valores são recebidos em um *array* de *Double*). A aplicação por sua vez, pode então comparar o número de atributos disponíveis (informado pelo *QoC Evaluator*) com o número de atributos que a informação deveria possuir (informado pelo *technology device*) e assim saber o grau de completude da informação.

- **Localização da fonte**: A localização da fonte pode ser inserida na informação de contexto pelo componente *technology device* (e.g. SensorPhone, Zephyr, SensorTag) (ver Figura 3.1), na camada M-Hub. Neste caso, o *QoC Evaluator* apenas grava esse valor no atributo *source location* da informação de contexto. Se a localização da fonte não estiver disponível, o *QoC Evaluator* insere a localização atual do *gateway* utilizando o serviço de localização do dispositivo onde o *middleware* está executando.

- **Intervalo de medição**: O intervalo entre as medições é calculado pelo *QoC Evaluator* como sendo o tempo decorrido entre o recebimento de cada amostra.

- **Resolução numérica**: A resolução numérica do dado é obtida a partir da contagem da quantidade de casas decimais contidas na medição.

- **Idade**: A idade é um parâmetro de QoC especial que é calculado dinamicamente no momento em que é utilizado. Ao requisitar a idade de determinada informação de contexto, o cliente recebe a informação calculada como a diferença entre o momento da requisição e o tempo de medição. Portanto, a idade da informação de contexto utilizada pela aplicação não é calculada pelo *QoC Evaluator*. No entanto, o *QoC Evaluator* calcula a idade da informação de contexto para poder calcular a média que será enviada aos serviços de diretório.

Além da qualidade associada a cada amostra da informação de contexto, o *QoC Evaluator* também calcula um valor médio das n últimas amostras recebidas de cada sensor (onde n é um parâmetro de configuração *QoC Evaluator*). Essa média serve como um indicador da qualidade de contexto do dispositivo que produziu as amostras, podendo ser usado como um valor/métrica de referência para aplicações que desejam selecionar os provedores de serviço com base na QoC ofertada. Por esse motivo, o valor médio de cada parâmetro de QoC das informações de contexto de cada sensor é enviada aos diretórios de serviços locais e global para que possa ser consultada pelas aplicações consumidoras locais e remotas.

O componente *QoC Evaluator* utiliza um Agente de Processamento de Eventos (EPA - *Event Processing Agent*) (ver Seção 2.4) que analisa o fluxo de dados vindos do S2PA. O objetivo desse EPA é detectar novos tipos de dados de contexto, bem como aplicar funções de agregação a fim de calcular a qualidade média das informações pertencentes a um mesmo tipo. Os eventos que resultam desse processamento geram um novo fluxo de eventos que são processados por este mesmo EPA. Este novo fluxo é utilizado pelo *Local Directory Service* para responder às consultas de contexto publicadas pelas aplicações, permitindo assim descobrir os provedores que atendem os requisitos de contexto e de QoC dos consumidores.

O componente *QoC Evaluator* é representado pela classe `QoCEvaluator` apresentada no diagrama de classes da Figura 3.7. Optou-se por mostrar também a classe `LocalDirectoryService` com a qual o *QoC Evaluator* interage através do EPA implementado em ambas as classes no atributo `epService`. Conforme implementado, os atributos `epService` nas duas classes referenciam o mesmo EPA em tempo de execução. Os métodos `void onMessageEvent()` são utilizados pelo *S2PA* da camada *M-Hub* para enviar os dados de contexto para o *Qoc Evaluator*.

Também podemos observar no diagrama os diversos métodos que calculam os valores dos parâmetros de QoC.

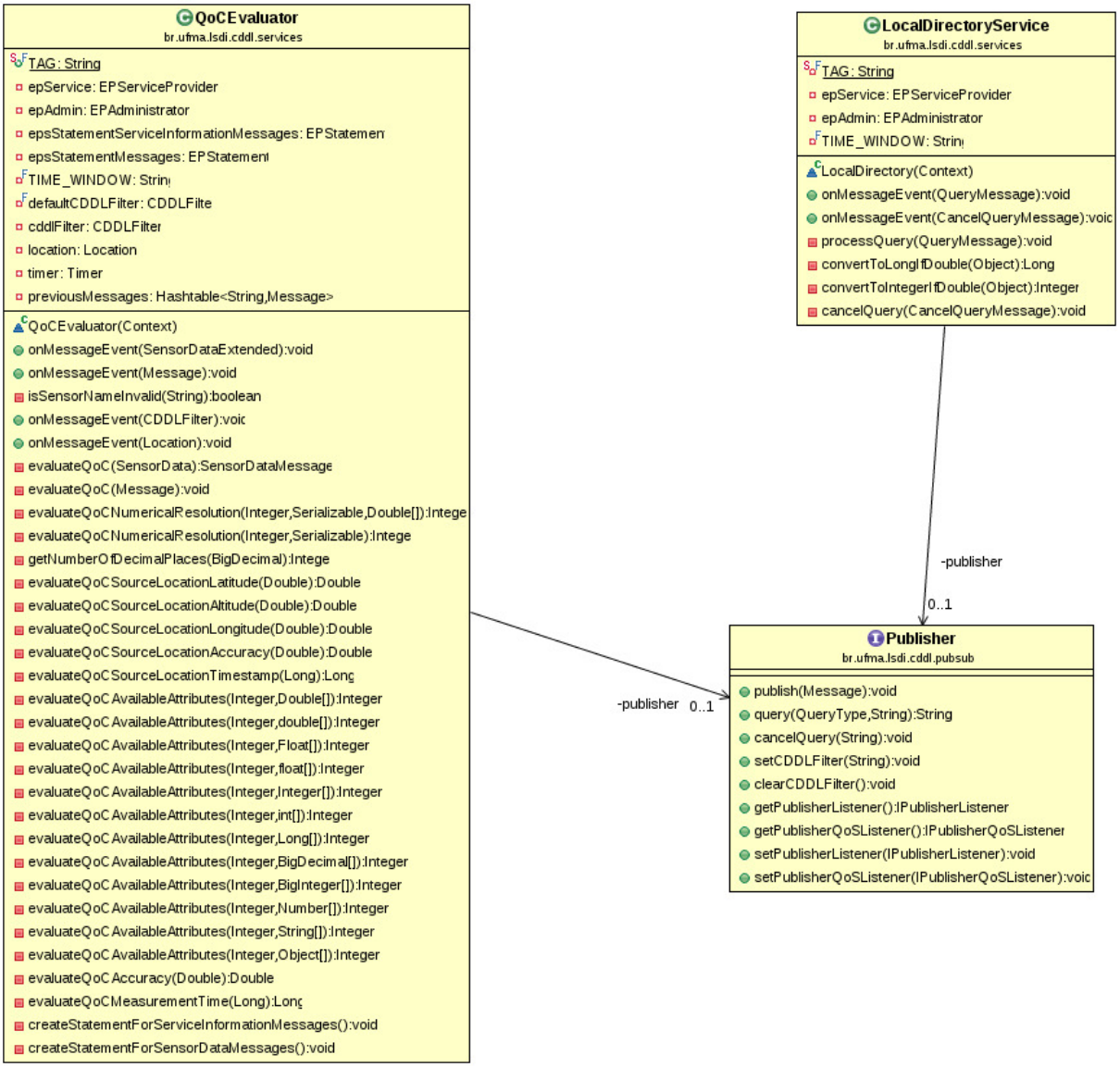

Figura 3.7: Diagrama de classes do QoC Evaluator

O diagrama de sequência apresentado na Figura 3.8 mostra o processo de avaliação da QoC e seu relacionamento com os demais componentes. O componente *S2PA* da camada M-Hub envia para o *QoC Evaluator* os dados de sensores (`SensorData`). A seguir o *QoC Evaluator* calcula os valores dos parâmetros de QoC que serão anexados à informação de contexto. Essa informação é então publicada utilizando-se o componente *Publisher*.

Além de anexar os valores de QoC calculados como metadados nas informações de contexto, o *QoC Evaluator* calcula a média de cada parâmetro de QoC dentro de um intervalo de tempo configurável utilizando para isso um EPA. A regra

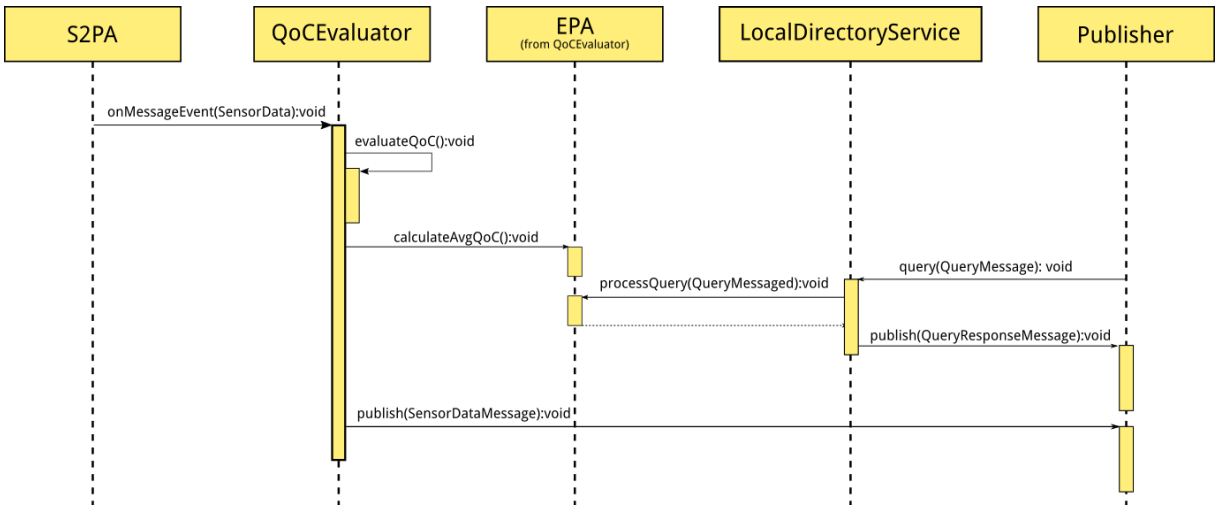

Figura 3.8: Diagrama de sequência do QoC Evaluator

é uma declaração EPL que calcula a média dos valores dos parâmetros de QoC é mostrada no listagem 3.5. Essa regra captura os eventos de novos serviços dentro de uma janela de tempo deslizante (configurável) cujo padrão é 5s e deriva novos eventos do tipo `ServiceInformationMessage`, que serão utilizados pelos serviços de diretórios para responder a consultas sobre as QoCs dos serviços disponíveis.

```
1 insert into ServiceInformationMessage (
2   publisherID, serviceName, accuracy, measurementTime, availableAttributes,
       sourceLocationLatitude, sourceLocationLongitude,
      sourceLocationAltitude, measurementInterval, numericalResolution, age
3 ) select
4   publisherID, serviceName, avg(accuracy), avg(measurementTime), avg(
      availableAttributes), avg(sourceLocationLatitude), avg(
      sourceLocationLongitude), avg(sourceLocationAltitude), avg(
      measurementInterval), avg(numericalResolution), avg(age)
5    from SensorDataMessage.win:time(TIME_WINDOW)
6    group by publisherID, serviceName;
```

Listagem 3.5: EPL para cálculo das médias das QoCs

3.4.2 Filter

O diagrama de classes na Figura 3.9 mostra a classe `Filter` e seus principais relacionamentos. Conforme pode ser observado, os componentes *Publisher* e *Subscriber* obtêm um referência para o filtro através do método `Filter getFilter()` da classe `Client`. De posse desta referência, eles podem utilizar o

método `void setFilter(String epl)` para criar uma regra, escrita na linguagem EPL, para o processo de filtragem, o método `void clear()` para remover o filtro e o método `boolean isSet()` para verificar se existe um filtro ativo.

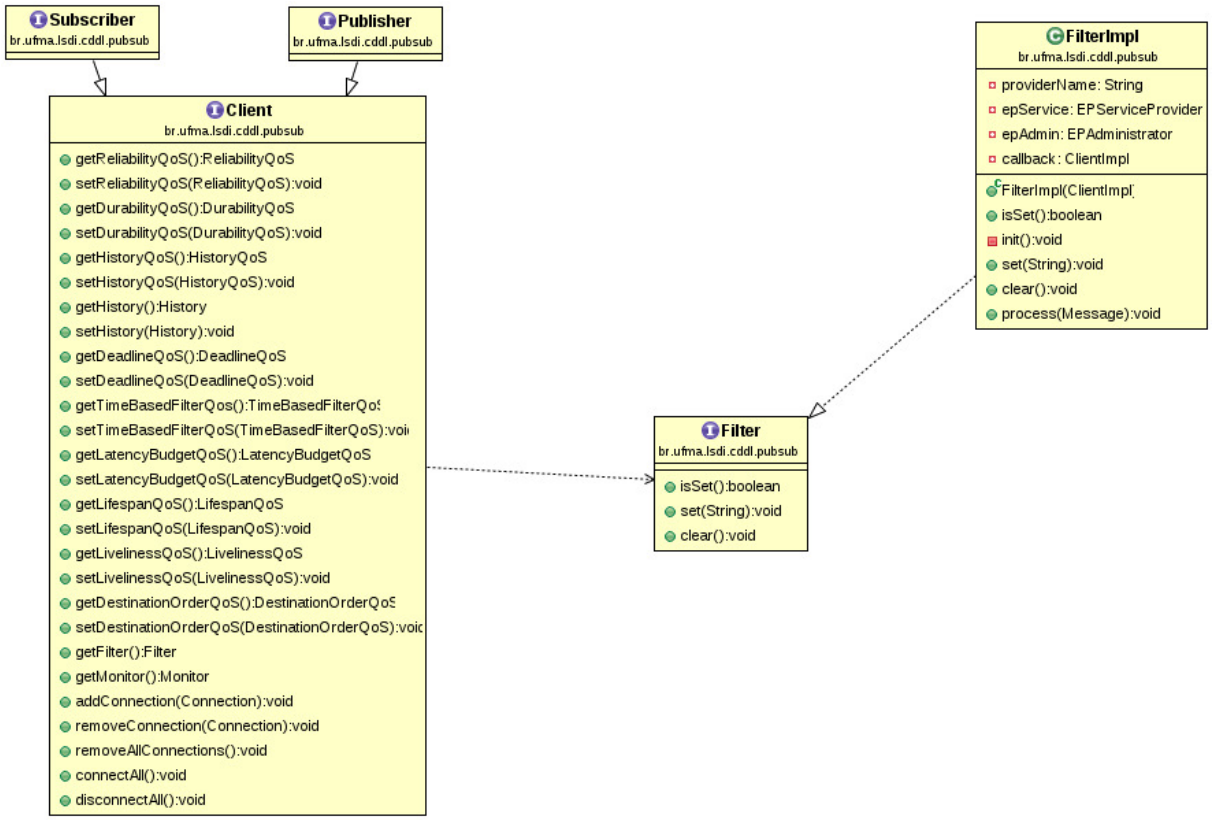

Figura 3.9: Diagrama de classes do Filter

A Figura 3.10, apresenta um diagrama de sequência onde podemos observar uma aplicação cliente incluindo um filtro no processo de subscrição de uma informação de contexto. Primeiramente a aplicação obtém uma referência para o filtro utilizando o método `Filter getFilter()`. Em seguida, especifica o filtro através do método `void set(String epl)` passando uma regra escrita na linguagem EPL para a filtragem. Neste momento, o componente *Filter* cria uma regra no seu EPA para processar as novas mensagens. Após especificar o filtro, a aplicação pode subscrever um serviço (no diagrama, utilizando o método `void subscribeServiceByName(String serviceName)`). Como existe uma regra de filtragem ativa, o componente *Subscriber* solicita ao *Filter* que este se subscreva no *Broker* no tópico de interesse da aplicação. À medida que as mensagens chegam ao *Broker*, ele as envia para o *Filter* que utiliza o EPA para processá-las. Após o processamento, as mensagens já filtradas são enviadas para o *Subscriber* que as repassa para a aplicação, finalizando o processo.

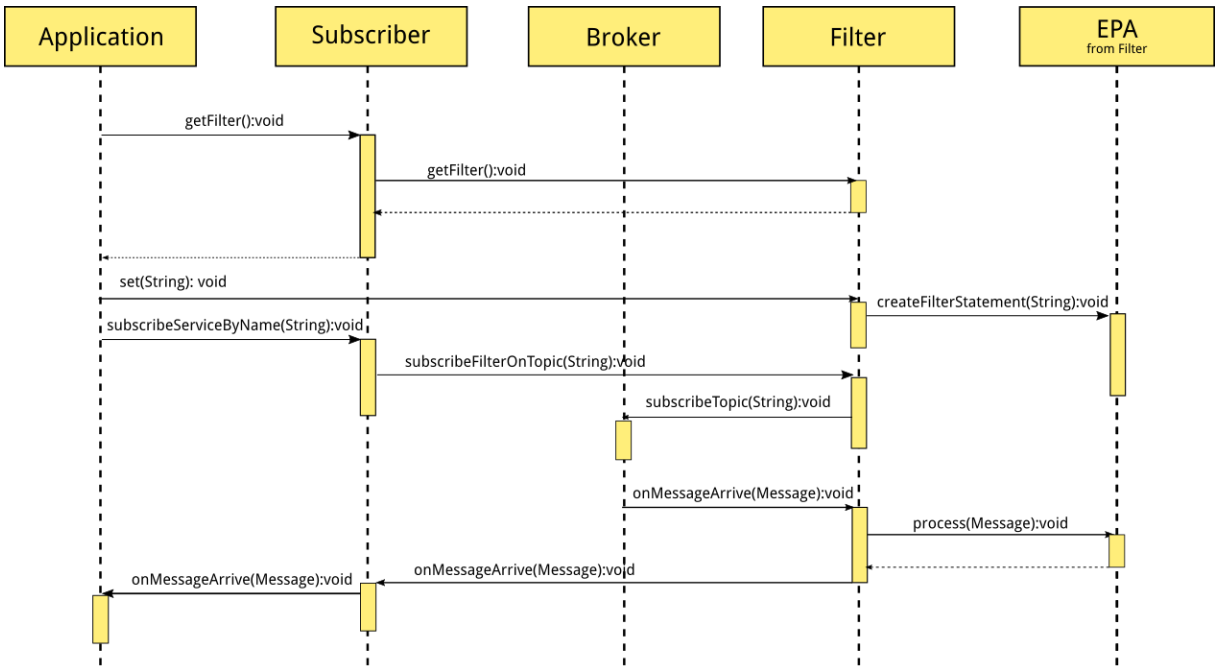

Figura 3.10: Diagrama de sequência do processo de filtragem

3.4.3 Monitor

O diagrama de classes na Figura 3.11 mostra a classe `Monitor` e seus principais relacionamentos. Os componentes *Publisher* e *Subscriber* obtêm um referência para o monitor através do método `Monitor getMonitor()` da classe `Client`. De posse desta referência, eles podem utilizar o método `String addRule(String epl)` para adicionar regras de monitoramento e seus respectivos *listeners* (classe `MonitorListener`) e o método `void removeRule(String ruleId)` para remover uma regra. Ao receber uma mensagem que satisfaça os critérios de um regra, o *Monitor* chama o método `void onEvent(Message message)` do *listener* daquela regra. O *listener*, por sua vez, deve ser implementado pela aplicação para reagir de acordo com seus interesses.

A Figura 3.12, apresenta um diagrama de sequência onde podemos observar uma aplicação cliente incluindo um monitoramento no processo de subscrição de uma informação de contexto. Primeiramente a aplicação obtém uma referência para o monitor utilizando o método `Monitor getMonitor()`. Em seguida, especifica a regra de monitoramento através do método `String addRule(String epl)` passando uma regra na linguagem EPL e o *listener* que receberá as notificações quando a regra for satisfeita. Neste momento, o componente *Monitor* cria uma regra no seu EPA para processar as novas mensagens. Após

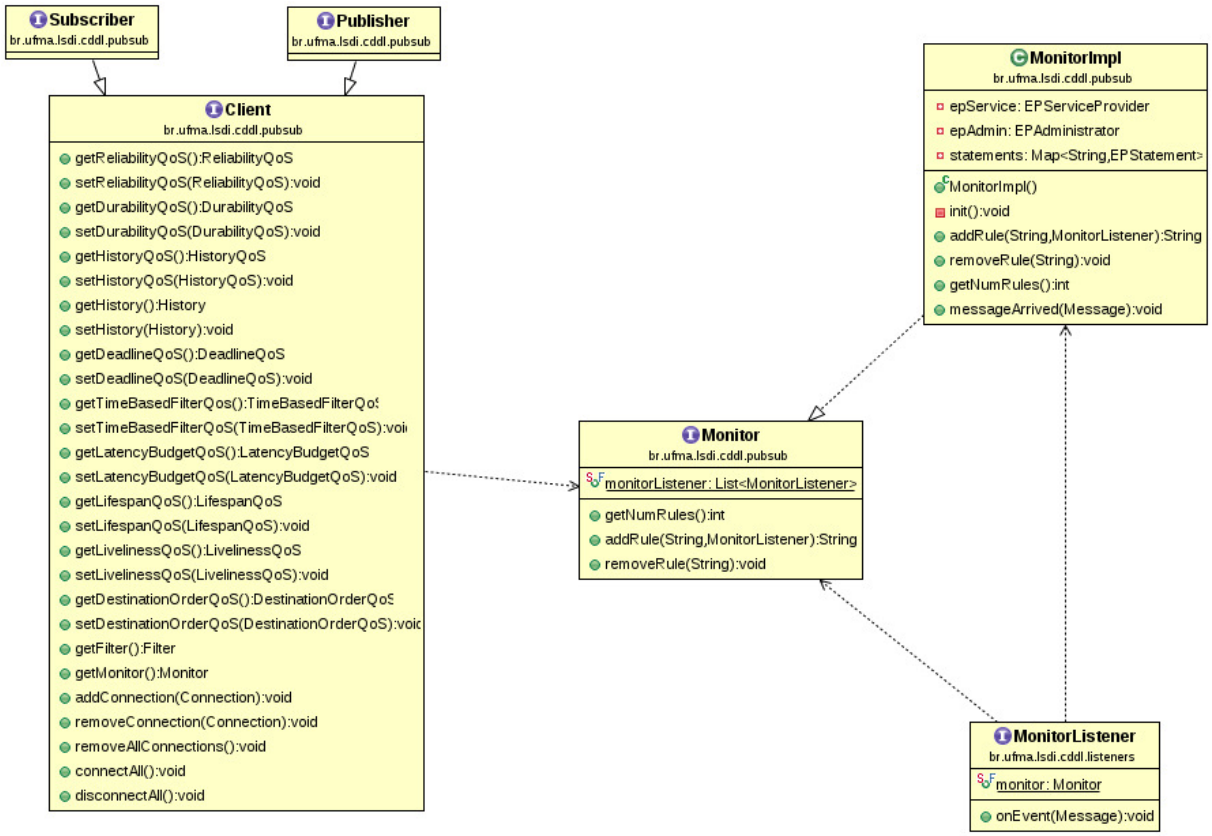

Figura 3.11: Diagrama de classes do Monitor

especificar a regra de monitoramento, a aplicação pode subscrever um serviço (no diagrama, utilizando o método `void subscribeServiceByName(String serviceName)`). Como existe uma regra de monitoramento ativa, o componente *Subscriber* solicita ao *Monitor* que este também se subscreva no *Broker* no tópico de interesse da aplicação. À medida que as mensagens chegam ao *Broker*, ele as envia para o *Monitor* que utiliza o EPA para processá-las. Durante o processamento, as mensagens que atendam à regra são enviadas para o *MonitorListener* definido pela aplicação. O monitoramento permanece ativo enquanto a regra não for removida do *Monitor*.

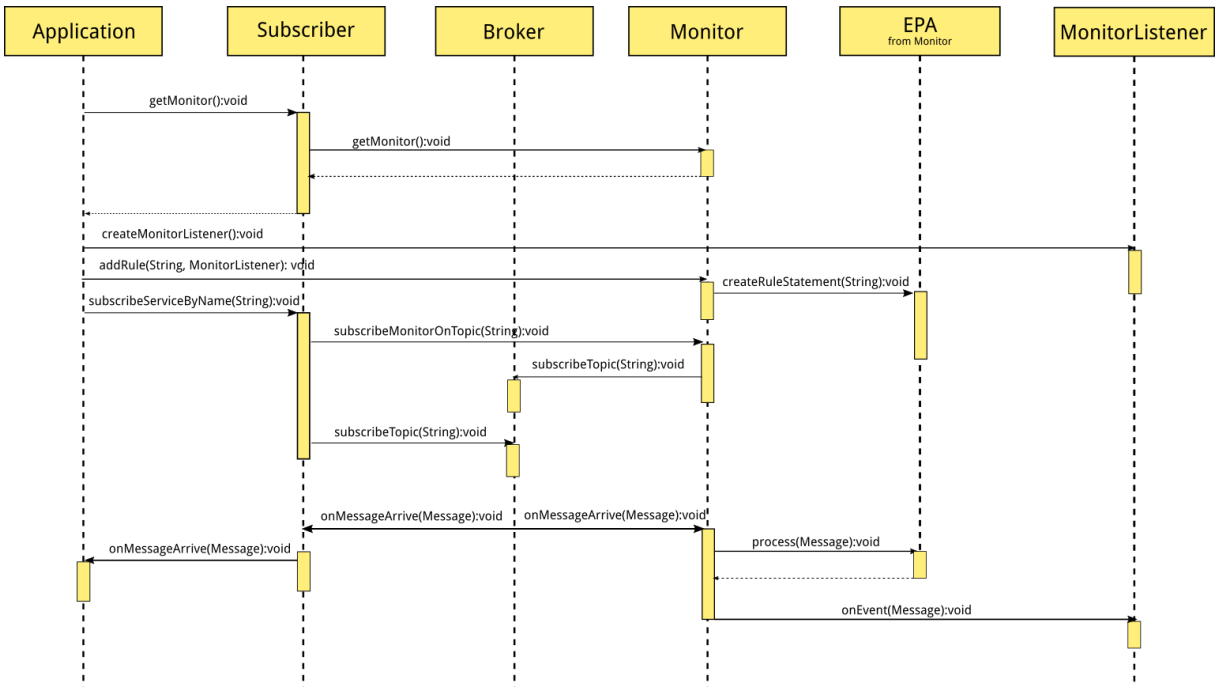

Figura 3.12: Diagrama de sequência do processo de monitoramento

4 Avaliação dos Mecanismos Propostos

4.1 Avaliação de Desempenho do Mecanismo de Monitoramento

A habilidade da aplicação consumidora das informações de contexto detectar rapidamente a alteração na QoC dos dados é muito importante já que o consumo de dados com qualidade inadequada considerando os requisitos da aplicação pode acarretar em um menor experiência de uso por parte do usuário ou a aplicação não funcionar adequadamente. Para avaliar o mecanismo de monitoramento da nossa solução, conduzimos um experimento com o objetivo de detectar o desempenho do *middleware* em relação ao tempo necessário para detecção da variação em um parâmetro de QoC.

Escolhemos duas métricas para serem avaliadas: a) o tempo em milissegundos que a aplicação leva para detectar uma mudança na QoC desde o momento em que ocorreu a alteração na qualidade da informação publicada e b) o tempo que a mensagem leva para ser processada pelo componente *Monitor* (ver Seção 3.2 para descrição dos componentes) utilizando o EPA.

Para a realização do experimento instalamos uma aplicação teste que utilizou o *middleware* em um *smartphone* LG 10K com 2 GB de memória RAM, processador quad-core 1.2 GHz e sistema operacional Android 6.0. Um notebook com processador Intel Core i7 2.5 Ghz com 16 GB de memória RAM, sistema operacional Ubuntu 14.04 LTS e interface de rede Wi-Fi 802.11 foi utilizado como *broker*. Ambos os equipamentos foram isolados em uma rede Wi-Fi própria com o notebook configurado como *access point* de modo a evitar ao máximo interferências de fatores externos no processo de avaliação.

A aplicação teste instanciou um componente *Publisher* que enviou mensagens para o *broker*. A mesma aplicação instanciou um componente *Subscriber* para subscrever as mensagens enviadas pelo *Publisher*. Optamos por essa configuração

para que o relógio utilizado para marcar os tempos fosse o mesmo e não precisarmos de um mecanismo de sincronização externo. A Figura 4.1 ilustra o fluxo de dados durante a execução da aplicação teste.

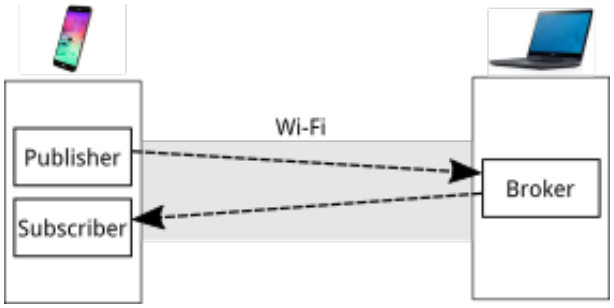

Figura 4.1: Fluxo de dados da aplicação teste

O experimento foi realizado como se segue: o *Publisher* publicou 1 mensagem por segundo durante 30 minutos. Essas mensagens foram geradas sinteticamente com uma acurácia com valor 1,0. Intercaladas a essas mensagens, a cada 10 segundos o *Publisher* publicou uma mensagem com acurácia com valor 0,5 para simular uma queda na acurácia da informação de contexto. A aplicação iniciou o mecanismo de monitoramento com a regra: `SELECT * FROM SensorDataMessage WHERE ACCURACY < 1`. Isso significa que a aplicação será notificada sempre que a acurácia tivesse um valor menor que 1. Executamos o experimento 5 vezes.

Paro o cálculo da métrica que indica o tempo que a aplicação leva para detectar uma mudança na QoC desde o momento em que ocorreu a alteração na qualidade da informação publicada, medimos a diferença entre o instante em que o componente *Publisher* publica a mensagem com a qualidade que satisfaz a regra de monitoramento e o instante em que o *listener* (informado pela aplicação quando esta adicionou a regra de monitoramento) recebe a notificação da alteração na qualidade da informação. Para o cálculo da métrica que indica o tempo que a mensagem leva para ser processada pelo componente *Monitor* utilizando o EPA, medimos a diferença entre o instante em que o componente *Monitor* recebe a mensagem vinda do componente *QoC Evaluator* e o instante em que o EPA utilizado pelo *Monitor* chama o *listener* que foi informado pela aplicação quando esta adicionou a regra de monitoramento.

A Tabela 4.1 apresenta os resultados da primeira métrica (tempo em milissegundos que a aplicação leva para detectar uma mudança na QoC) obtidos em cinco execuções sucessivas do experimento. Conforme podemos observar, em um

ambiente de rede local como foi o caso do experimento realizado, a aplicação de teste foi notificada da variação na QoC em média em 105,25 ms.

	1° experimento	2° experimento	3° experimento	4° experimento	5° experimento	Média	Desvio Padrão	Intervalo de Confiança (95%)
Tempo em ms	106,25	104,22	104,23	105,98	105,59	105,25	0,97	104,41 – 106,09

Tabela 4.1: Tempo que a aplicação leva para detectar uma mudança na QoC

A Tabela 4.2 apresenta os resultados da segunda métrica (tempo em milissegundos que a mensagem leva para ser processada pelo componente *Monitor*) obtidos em cinco execuções sucessivas do experimento. Conforme podemos observar, o tempo de processamento do serviço de monitoramento é de 6,33 ms em média, considerando-se a publicação de 1 mensagem a cada segundo.

	1° experimento	2° experimento	3° experimento	4° experimento	5° experimento	Média	Desvio Padrão	Intervalo de Confiança (95%)
Tempo em ms	6,56	6,48	6,15	6,59	5,88	6,33	0,30	6,06 – 6,60

Tabela 4.2: Tempo de processamento da mensagem pelo *Monitor*

É importante destacar que foram geradas 180 mensagens contendo a acurácia inferior a 1,0 e que, em todos os casos, o mecanismo de monitoramento detectou esta variação e notificou a aplicação de forma apropriada. Ou seja, não houve casos em que a variação ocorreu e aplicação não foi informada.

Concluímos, a partir deste experimento, considerando-se o ambiente de uma rede local Wi-Fi 802.11, que o tempo médio que a aplicação leva para receber as notificações de variações na QoC (em média 105,25 ms) é adequado para que a mesma possa reagir prontamente a variações na QoC. Com relação ao tempo que o mecanismo de monitoramento leva para processar cada mensagem, considerando uma mensagem a cada segundo, concluímos que tal tempo de processamento (em média 6,33 ms) é rápido o suficiente para não degradar o desempenho da aplicação.

4.2 Avaliação do Consumo de Memória dos Mecanismos

O objetivo deste experimento foi medir a quantidade de memória alocada para os componentes do *middleware* responsáveis pelo monitoramento e avaliação de QoC, ambos baseado na utilização de agentes de processamento de eventos complexos

Foi utilizada a mesma aplicação teste empregada no experimento anterior, variando-se apenas a taxa de publicação, que neste experimento foi de 1 mensagem por segundo (uma frequência considerada moderada). A aplicação foi executada durante 30 minutos. A partir do uso da ferramenta Android Monitor (embutida na IDE Android Studio) foram coletados dados relativos ao consumo de memória em 5 instantes de execução da aplicação: 5, 10, 15, 20 e 25 minutos. A tabela 4.3 exibe a média resultados obtidos considerando 5 repetições.

	5 min	**10 min**	**15 min**	**20 min**	**25 min**	**Média**	**Desvio Padrão**
QoC Evaluator	263	262	268	269	270	266	2.77
Monitor	213	213	216	219	280	214	3.14

Tabela 4.3: Consumo de Memória em Kbytes

A partir desses resultados é possível observar que os componentes de monitoramento e avaliação de QoC necessitam de pouca quantidade de memória para processar fluxos de dados com frequência moderada.

4.2.1 Avaliação do Consumo de Bateria

O objetivo deste experimento foi avaliar o consumo de bateria por parte dos componentes do *middleware*. Para isso foi utilizada a mesma aplicação teste do experimento anterior e igual taxa de publicação de dados. A aplicação foi executada durante 12 horas, sendo que o nível da bateria foi medido antes e depois do tempo de execução da mesma. É importante destacar que durante a execução da aplicação a tela do dispositivo permaneceu desligada. Além disso, com a exceção dos processos do sistema operacional, não haviam outros processos executando no dispositivo. O consumo de bateria medido foi de 15.6 % após as 12 horas.

	Mínimo	**Máximo**	**Média**	**Desvio Padrão**
Consumo de Bateria	15 %	16 %	15.6 %	0.55

Tabela 4.4: Consumo de Bateria

Considera-se que esse consumo é baixo, indicando que o *middleware* pode ser utilizado para publicar dados e monitorar QoC em dispositivos móveis de uso convencional.

5 Trabalhos Relacionados e Análise Comparativa

Apesar do desenvolvimento de *middleware* de contexto ser uma área de grande interesse, e que várias propostas tenham surgido, poucas delas focaram nos mecanismos de avaliação de QoC, particularmente em prover um mecanismo abrangente que envolva tanto QoS quanto QoI. O cenário tampouco conta com muitos trabalhos que foquem o monitoramento de QoC. Perera et al. [69] e Bandyopadhyay et al. [4] analisarem vários tipos de *middleware*, mas sem levar em consideração a QoC como critério comparativo. No melhor do nosso conhecimento, as propostas de *middleware* com suporte a QoC que mais se destacam são: QoMonitor [5], AWARENESS [79], COSMOS [1, 20], COPAL [50], INCOME [3, 60, 61] e SALES [22, 24, 25, 32]. Essas propostas serão apresentadas a seguir.

5.1 QoMonitor

QoMonitor [5] é um sistema de monitoramento de metadados que recebe requisições síncronas e assíncronas de clientes (aplicações e/ou *middleware* ubíquos), recupera metadados de provedores de contexto e os envia aos clientes. Utilizando o *middleware*, aplicações ubíquas podem se concentrar nos problemas do negócio e abstrair as complexidades relacionadas com o monitoramento de metadados. Os metadados monitorados pelo QoMonitor ficam disponíveis para as aplicações. O QoMonitor pode ainda ser associado a um *middleware* responsável por gerenciar essas informações de modo a selecionar os serviços que serão usados pela aplicação, por exemplo.

O QoMonitor é constituído de três repositórios: a) um *Metadata Repository*, que armazena todos os metadados de QoS e QoI dos serviços monitorados e os metadados fornecidos pelas provedores dos serviços; b) um *Service Repository*, que armazena informações sobre os serviços monitorados e os parâmetros necessários para se comunicar com eles e c) um *Client Repository*, que armazena informações sobre os

clientes de modo a permitir a comunicação com os clientes. Além disso, o QoMonitor contém: a) um *Ontology Module*, que é responsável por especificar metadados usando um modelo de ontologia que representa os conceitos de modo não ambíguo; b) um *Request Handler*, que recebe requisições dos clientes, coleta os metadados e os envia para estes clientes e c) um *Assesment Module*, que é responsável por efetivamente monitorar e avaliar metadados de QoS/QoI dos serviços armazenados no repositório de serviços.

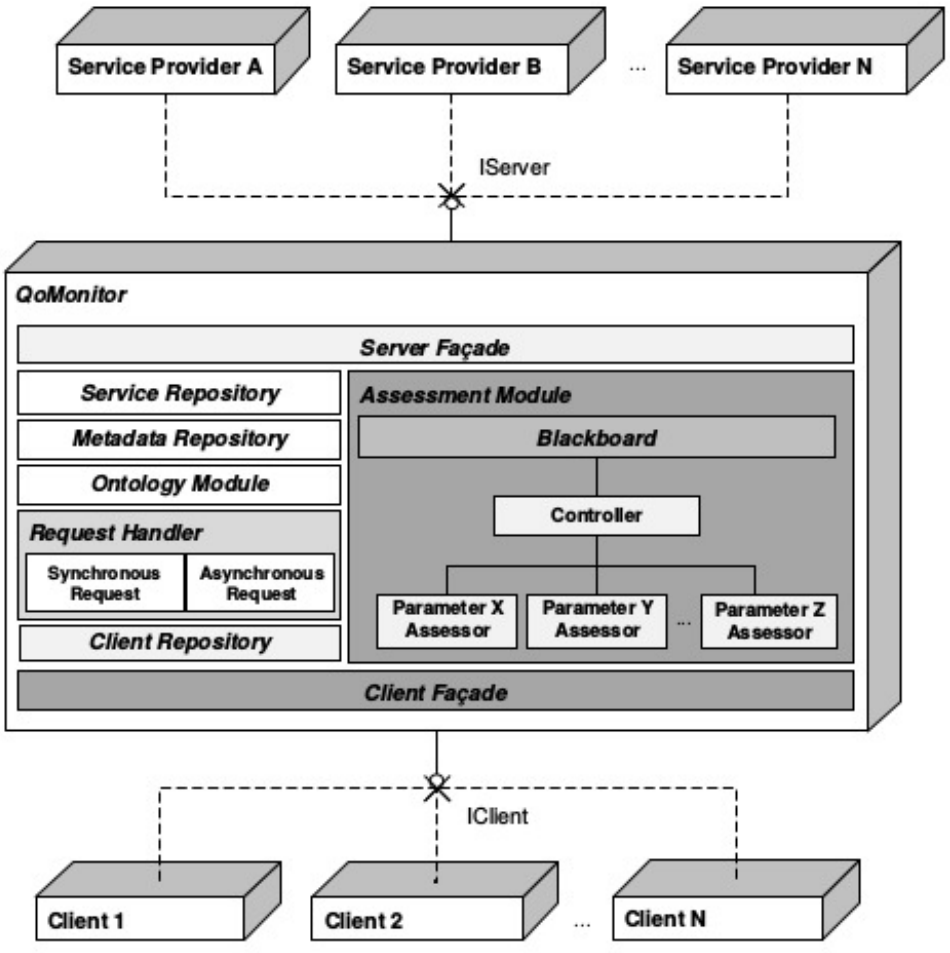

Figura 5.1: Arquitetura Geral do QoMonitor

A Figura 5.1 ilustra a arquitetura do QoMonitor. O QoMonitor fornece duas interfaces de comunicação: IClient para comunicação com clientes e IServer para se comunicar com os provedores de serviços. A *Server Façade* é responsável por registrar novos serviços no repositório de serviços e se comunicar com os provedores. O *Service Repository* é responsável por armazenar informações sobre todos os serviços

monitorados e os parâmetros necessários para se comunicar com eles. A *Client Façade* é responsável por permitir a comunicação dos clientes com o monitor, que pode ser qualquer aplicação ou *middleware* ubíquo que precisa fazer uso de parâmetros de QoS/QoI. Os clientes podem se registrar no monitor e fazer solicitações síncronas e assíncronas. Ao executar solicitações assíncronas, o cliente deve implementar um método de *callback* para permitir a comunicação entre o monitor e o cliente, de modo que esse método seja responsável por receber a resposta do monitor em relação à solicitação assíncrona. O monitor verifica periodicamente a condição de retorno e, se for satisfeita, responde ao cliente fornecendo os parâmetros com seus respectivos valores representados na forma da ontologia. O *Metadata Repository* é responsável pela persistência de todos os metadados de QoS/QoI avaliados pelo monitor e também dos metadados de QoS/QoI fornecidos pelos provedores de serviços. Por sua vez, o *Ontology Module* é responsável por representar esses dados sob a forma de uma ontologia.

O componente principal do monitor é o *Assesment Module* que é responsável por avaliar os metadados de QoS/QoI dos serviços armazenados no *Service Repository* e monitorá-los e é composto por três tipos de elementos: assessores, *Blackboard* e *Controller*. Cada assessor é responsável por avaliar um parâmetro de qualidade (QoS/QoI) específico a partir de informações coletadas através de solicitações aos serviços monitorados pelo *Assesment Module*. Esta informação é: (i) o tempo gasto para completar o pedido (*CompletedTime*); (ii) se o serviço estava disponível ou não (*isAvailable*); (iii) o instante em que o pedido foi feito (*TimeStamp*), e; (iv) a data e a hora de criação/detecção das informações de contexto fornecidas pelo serviço que é utilizada para inferir a idade (*Age*) das informações de contexto fornecidas pelo serviço. O componente *Blackboard* incorpora a ideia de um repositório de dados compartilhado usado pelos avaliadores de diferentes parâmetros QoS/QoI para calcular o valor desses parâmetros. O componente *Controller* é responsável por controlar o acesso às informações armazenadas no quadro-negro e as informações obtidas a partir da avaliação dos parâmetros, para que os avaliadores não conheçam a fonte dos dados que eles usam para fazer a avaliação, modularizando assim a arquitetura. O monitoramento de serviços funciona de forma independente, usando *threads* e começa no momento em que o monitor está disponível, de modo que as operações de monitoramento e avaliação sejam executadas enquanto o monitor recebe

e responde solicitações. Este monitoramento contínuo destina-se a acelerar o tempo de resposta dos pedidos, pois, quando uma solicitação é feita, os metadados de QoS/QoI já estão armazenados e podem ser acessados pelo *Request Handler* para responder aos clientes. O *Request Handle* é responsável por recuperar metadados de QoS/QoI e encaminhá-los aos clientes. Quando um cliente faz uma solicitação síncrona, a *Client Façade* encaminha-a para o *Request Handler*, que recupera os dados atuais e responde à *Client Façade*. Quando uma solicitação assíncrona é encaminhada para o *Request Handler*, ele monitora se os dados de QoS/QoI satisfazem a condição de retorno informada pelo cliente; Neste caso, o *Request Handler* monitora continuamente esses dados e, quando a condição de retorno é atendida, responde imediatamente à *Client Façade* que chama o método de *callback* implementado pelo cliente.

Em relação ao suporte à QoC, o QoMonitor implementa poucos parâmetros (apenas quatro: *completed time, is available, timestamp e age*). O serviços podem informar seus próprios metadados de QoS/QoI que são repassados para os clientes mas estes metadados não são utilizados como parâmetros para o monitoramento realizado pelo *Request Handler*, sendo apenas inseridos nas informações de contexto. Em relação ao monitoramento, o *middleware* fornece um monitoramento contínuo que é iniciado quando o cliente faz uma solicitação assíncrona ao *Request Handler* e passa um método de *callback* para ser executado quando a condição de monitoramento especificada pela aplicação for satisfeita. A aplicação fica responsável por implementar as funcionalidades para lidar com as respostas da solicitação enviada pelo serviço de monitoramento.

5.2 AWARENESS

AWARENESS [79] é uma infraestrutura para o desenvolvimento de aplicações móveis cientes de contexto adaptativas. Além da distribuição, processamento e armazenamento, o *middleware* oferece também mecanismos de descoberta de serviços. Outra característica é o suporte à especificação de políticas de privacidade, definidas pelo usuário, que adaptam a QoC e o acesso à informação.

A arquitetura geral do AWARENESS é ilustrada por meio da Figura 5.2. O suporte ao desenvolvimento de aplicações do *middleware* provê um conjunto de APIs

e componentes distribuídos em duas camadas: a camada de aplicação e a camada de infraestrutura. Essas camadas são descritas a seguir. Na camada de aplicação estão as aplicações móveis cientes de contexto e pró-ativas, divididas em componentes clientes e servidores que podem interagir entre si e com a infraestrutura. A camada de infraestrutura dá suporte à execução das aplicações. Ela concentra funções de gerenciamento de contexto distribuído, inferência, registro, descoberta de serviços e persistência. Essa camada compreende também sensores específicos da infraestrutura. Esses sensores fornecem dados que serão distribuídos para as aplicações clientes. A heterogeneidade do hardware e de protocolos de comunicação é encapsulada por componentes envoltórios (*wrappers*). De acordo com os autores, a camada de infraestrutura pode usar internamente qualquer tecnologia de rede, com ou sem fio, para comunicação. Entretanto, nenhum exemplo de protocolo utilizado em testes é descrita no trabalho.

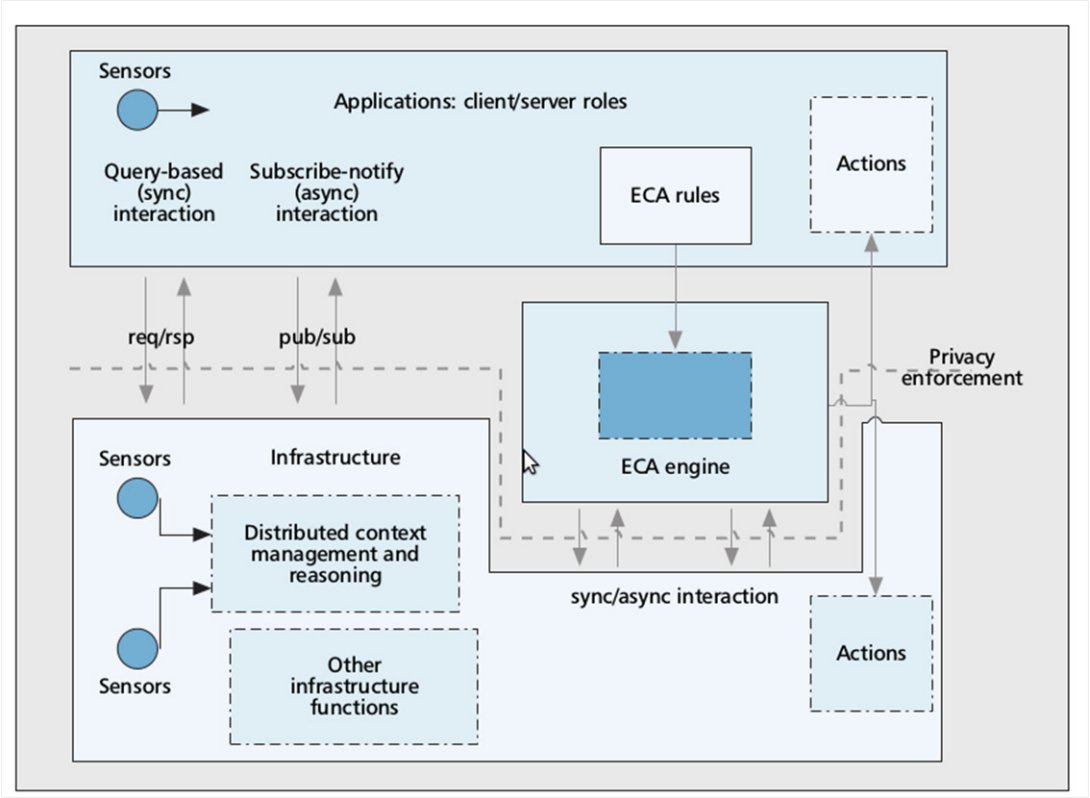

Figura 5.2: Arquitetura Geral do AWARENESS [79]

A arquitetura do CMS (*Context Managment Service*) é ilustrada por meio da Figura 5.3. O serviço de gerenciamento de contexto da infraestrutura provê duas interfaces principais: a *Context Broker* e a *Context Source*, que são descritas a seguir. A interface *Context Broker* fornece os métodos que permitem o registro e a

descoberta dos tipos e fontes de contexto disponíveis. O tipo de informação que pode ser registrado consta em uma ontologia particular. O mecanismo de descoberta de serviço pode operar de dois modos: ativo e passivo. No primeiro, a aplicação faz uma consulta e recebe imediatamente uma resposta contendo uma lista com serviços disponíveis. No modo passivo, a aplicação publica a consulta e se subscreve para receber a resposta posteriormente. A resposta virá assim que for encontrada a primeira correspondência positiva entre o exigido e o disponível. A interface *Context Source* fornece os métodos para recuperação de informação, que pode ser assíncrona (baseada em eventos) ou síncrono. No modo assíncrono, o cliente se subscreve para receber notificações de mudança do valor de uma informação de contexto em particular. No modo síncrono, o cliente realiza uma consulta que retorna o valor atual da informação de contexto. O *middleware* suporta consultas especificadas em SQL (*Structured Query Language*) e RDQL (*RDF Data Query Language*) [17]. O comportamento das aplicações pode ser definido por um conjunto de regras *Evento-Condição-Ação* que são executas na infraestrutura por uma *Engine ECA* que pode ser acessada pelas aplicações de maneira síncrona ou assíncrona utilizando um serviço de controle denominado ECA-CS. Quando os eventos de interesse do consumidor ocorrem, a infraestrutura dispara uma ação que será executada pela aplicação em resposta às mudanças de contexto. O *middleware* fornece ainda mecanismos de persistência, histórico e consulta, que são implementados pelo componente *Context Storage Service*. A implementação é baseada em um banco de dados relacional que oferece tanto acesso direto como uma interface para execução de inferência sobre os dados baseada em Jena[1].

Em relação ao suporte a QoC, as fontes de contexto podem especificar metadados que descrevem qualidade das informações. Os parâmetros suportados são: *Accuracy*, *Probability of Correctness*, *Trustworthiness* e *Up-to-dateness*. Os mecanismos de privacidade do *middleware* são baseados em políticas especificadas pelo usuário que definem as informações que podem ser compartilhadas, os destinatários e a QoC. Em alguns casos a política definida requer o consentimento direto do usuário, que libera a informação caso-a-caso.

[1] `https://jena.apache.org/tutorials/rdf_api.html`

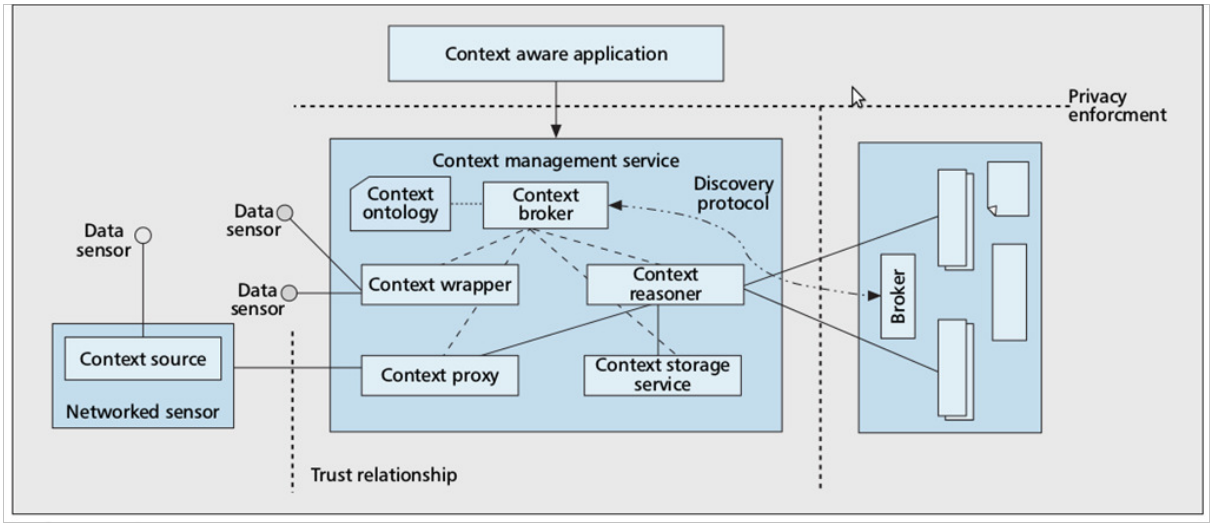

Figura 5.3: Componentes do CMS do AWARENESS [79]

5.3 COSMOS

COSMOS[2] (*COntext entitieS coMpositiOn and Sharing*) [20] é um framework para gerenciamento de contexto em ambientes ubíquos. O suporte ao desenvolvimento de aplicações cientes de contexto adaptativas utiliza um modelo de programação baseado em componentes Java que podem ser descritos/compostos utilizando-se uma DSL (*Domain Specific Language*) [35]. A comunicação entre os componentes é orientada a mensagens (MOM) e foi implementada utilizando a biblioteca DREAM [49].

Todos os componentes do COSMOS são implementados a partir de uma estrutura básica denominada *Context Node*, cuja arquitetura está ilustrada na Figura 5.4. Os nodos de contexto são organizados em hierarquias. A comunicação dentro da hierarquia de nodos de contexto pode ser de baixo para cima (notificação) ou de cima pra baixo (observação). Para o envio de notificações usa-se a interface *Push*, enquanto que para o recebimento de observações usa-se a interface *Pull*. Observações e notificações contêm informações de contexto encapsuladas, que também são modeladas utilizando nodos de contexto. O modelo de programação dos nodos de contexto é baseado em fractais e foi implementado usando a biblioteca Julia [11].

A arquitetura do COSMOS é conceitualmente dividida em três camadas: inferior, intermediária e superior. A camada inferior do COSMOS define a noção de *Context Collector*, que são nodos de contexto responsáveis por coletar dados de contexto de baixo nível a partir da interação com sistemas operacionais, redes e

[2] `http://picolibre.int-evry.fr/projects/cosmos`.

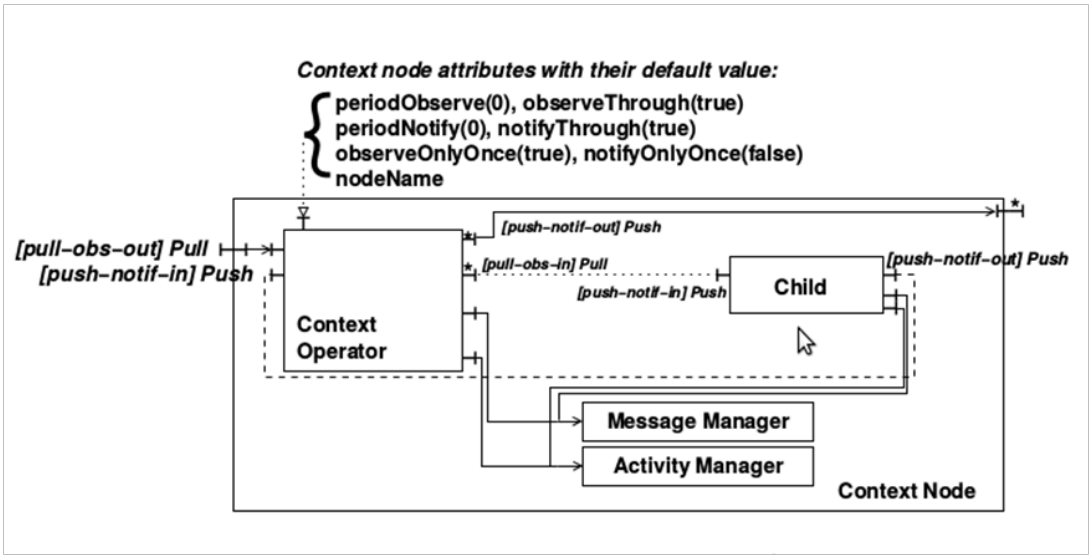

Figura 5.4: Arquitetura do *Context Node* do COSMOS [1]

sensores. Esses coletores também podem obter informações que representam as preferências dos usuários. Portanto, o coletor de contexto esconde das demais camadas a heterogeneidade e os protocolos de comunicação com as fontes de contexto. O suporte a aquisição de dados COPAL foi implementado com uso pelo framework SAJE [26]. A camada intermediária define a noção de processador de contexto. Cada *Context Processor* é um nodo de contexto que filtra e agrega os dados provenientes dos coletores de contexto a fim de obter informações de contexto de alto nível. Na camada superior estão os nodos de contexto responsáveis por inferir situações a partir dos dados coletados e processados nas camadas inferiores. As situações inferidas podem desencadear algum tipo de adaptação da aplicação. Segundos os autores, as técnicas de processamento e inferência a serem utilizadas dependem do domínio de cada aplicação. Por isso o *middleware* não oferece suporte a uma técnica específica de processamento, tais como processamento de eventos.

Abid et al. [1] estenderam a arquitetura do COSMOS a fim de adicionar o componente *QoC Context Node*. Esse componente é responsável por processar os dados de contexto coletados a fim de extrair e avaliar a QoC. A arquitetura do *QoC Context Node* é ilustrada na Figura 5.5.

O *QoC Context Node* é um nodo de contexto composto. Ele possui diversos coletores de contexto e outros subcomponentes que tratam da QoC. A extração e avaliação da qualidade dos dados/sensores é feita pelo subcomponente *QoC Operator*, cuja arquitetura é ilustrada na Figura 5.6.

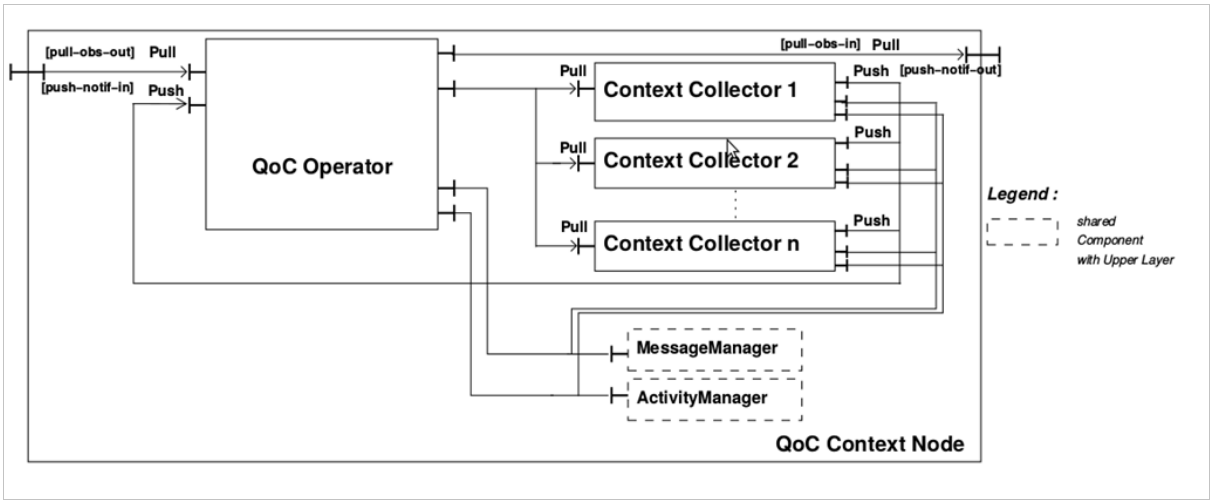

Figura 5.5: Arquitetura do *QoC Context Node* do COSMOS [1]

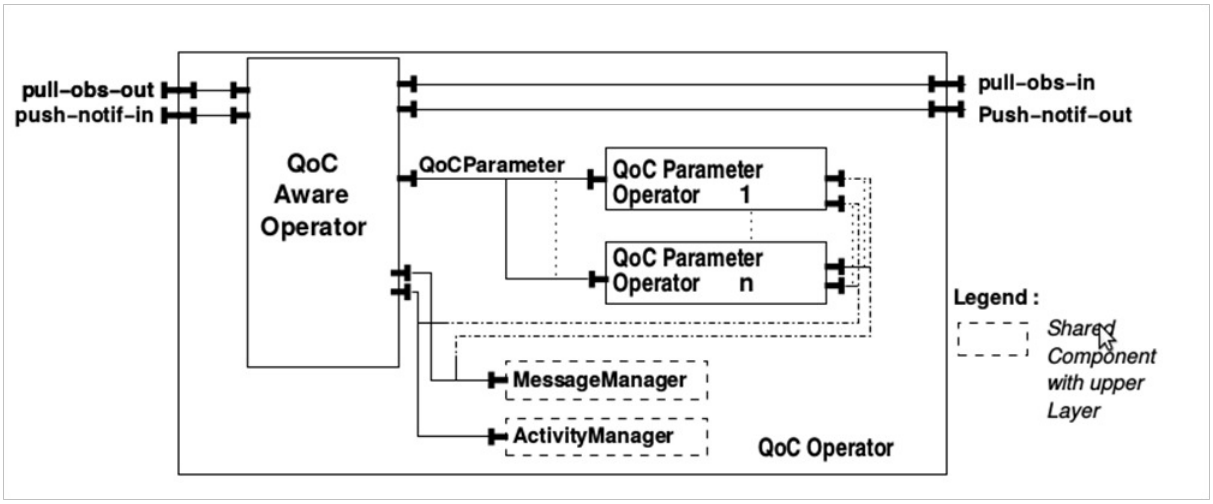

Figura 5.6: Arquitetura do *QoC Operator* do COSMOS [1]

Os parâmetros de QoC suportados pelo COSMOS são: *UpToDateness*, *Trustworthiness*, *Accuracy*, *Precision*, *Security* e *Completeness*. Entretanto os autores do trabalho afirmam que outros parâmetros podem ser facilmente adicionados. Cada parâmetro de QoC do COSMOS é computado separadamente pelo seu respectivo *QoC Parameter Operator*. Após serem calculados, os parâmetros de QoC são enviados para o subcomponente *QoC Aware Operator*, que é responsável pela transmissão das informações de QoC para as camadas superiores. Esse componente realiza a distribuição das informações de dois modos: *QoC embutida na informação* e *QoC separada da informação*. No modo *QoC embutida na informação*, todas as amostras da informação de contexto são enriquecidas com QoC. Esse modo é útil para aplicações interessadas tanto na informação de contexto em si quanto na QoC associada a ela. Este modo permite a filtragem das informações com base na QoC, apesar disto impor um maior

custo computacional. No modo *QoC separada da informação*, a QoC é enviada em mensagens separadas, periodicamente (*modo pull*) ou mediante uma requisição da aplicação (*modo push*). A informação sobre a qualidade de contexto enviada pode corresponder à última QoC computada ou a uma média dos últimos valores. O envio da QoC separadamente, além de consumir menos recursos, garante a compatibilidade com aplicações que usam versões do COSMOS que não suportam QoC.

A DSL utilizada pelo COSMOS é um subconjunto da linguagem FractalADL (*Architecture Description Language*) [48]. Entretanto, essa DSL ainda não suporta especificações de QoC. Segundo os autores, isso aconteceria em trabalhos futuros.

5.4 COPAL

COPAL (*COntext Provisioning for ALl*) [50] [3] é um *middleware* de contexto implementado em Java voltado ao desenvolvimento de aplicações adaptativas. Para isso, o COPAL fornece suporte a especificação e processamento (e.g, filtragem, agregação) de eventos que refletem as mudanças de contexto no ambiente de execução da aplicação. Ações específicas definidas pelas aplicações podem ser executas pelo *middleware* diante da ocorrência de determinados eventos. COPAL está inserido no contexto de desenvolvimento do projeto SM4ALL (*Smart Home for All*).

O suporte ao desenvolvimento de aplicações do COPAL provê uma linguagem para a descrição de componente chamada de COPAL-DSL. Posteriormente em [74] foi desenvolvida a linguagem COPAL ML (COPAL *Macro Language*), que estende a linguagem Java com algumas palavras-chave. Segundo os autores, a COPAL-ML esconde a complexidade e reduz o código necessário para o desenvolvimento de aplicações. A intenção é usar a COPAL-ML para escrever a lógica dos componentes, enquanto que a COPAL-DSL descreveria a sua implementação. Opcionalmente, se pode programar os componentes usando a COPAL API.

A arquitetura do COPAL, ilustrada na Figura 5.7, está dividia em três camadas: serviços de dispositivos, núcleo COPAL e serviços cientes de contexto.

A camada **Serviços de Dispositivos** descreve os dispositivos e sensores com os quais o COPAL interage. A heterogeneidade do hardware e dos protocolos

[3] `http://www.infosys.tuwien.ac.at/m2projects/sm4all/copal/downloads.html`

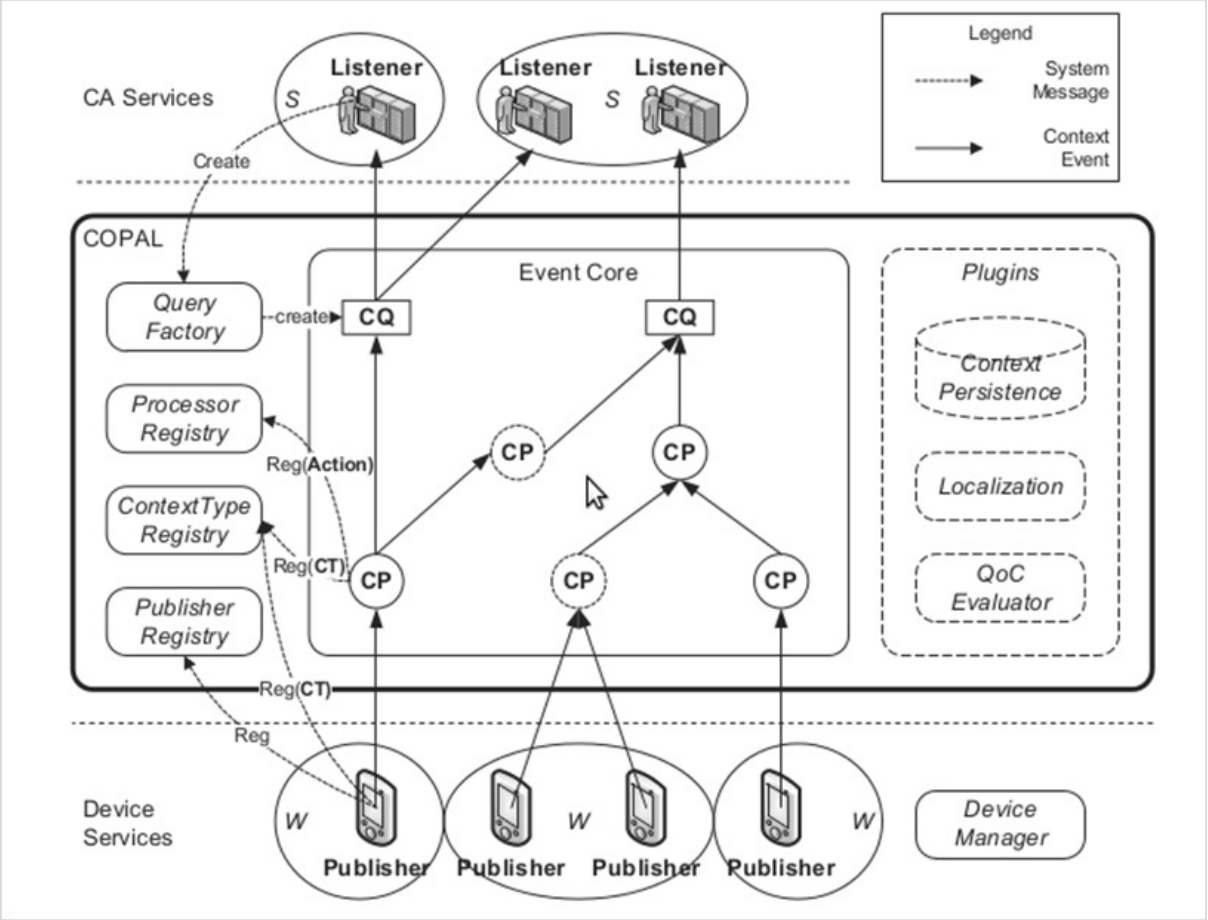

Figura 5.7: Arquitetura do COPAL [50]

de comunicação é escondida por (*wrappers*) que implementam os protocolos de acesso específicos de cada dispositivo e os expõem como serviços web para a camada superior. As descrições de serviço são feitas conforme o padrão UPnP [4](*Universal Plug and Play*). Do ponto de vista do COPAL, cada dispositivo é um *Publisher*. O serviço *Device Manager* armazena informações de hardware, mantém um catálogo de dispositivos e monitora o status de cada um.

A camada **Núcleo COPAL** registra componentes, realiza processamento e executa ações. Os serviços *Publisher Registry* e *Context Type Registry* são responsáveis pelo registro dos publicadores e tipos de informação de contexto (*Context Type*, respectivamente. O *Context Processor* tem a função de processar os tipos de contexto. Esse processamento pode resultar na execução de uma ação a qual ele está vinculado. Cada *Context Processor* deve ser registrado junto ao *Context Process Registry*.

[4]`http://www.upnp.org/`

A camada **Serviços Cientes de Contexto** contém *listeners* que as aplicações usam para que sejam notificadas quando suas consultas de contexto são atendidas. Uma consulta de contexto é identificada pelo seu nome e a declaração contém um tipo de contexto e um conjunto de critérios que permitem filtrar os dados. Cada *Listener* está associado a uma consulta, que pode ser reutilizada por vários *listeners*. *Query Factor* é componente de geração das consultas.

A troca de dados de contexto entre os *Publishers*, os *Listeners* e *ContextProcessors* é realizada pelo *Event Core Service* implementado com a Engine Esper CEP (*Complex Event Processing* [5]. Eventos em COPAL são documentos XML que estão em conformidade com os tipos de contexto, também definidos no esquema XML. Consultas no COPAL são transformadas para EPL *Event Processing Language* [13].

Eventos no COPAL possuem quatro atributos essenciais dos eventos de contexto são: *Source ID*, *Timestamp*, *Priority* e *Time To Live (TTL)*. *Source ID* e *Timestamp* são inseridos pelos publicadores da informação de contexto quando o evento é criado. A *Priority* (Prioridade) é utilizada para garantir que eventos de emergência sejam processados primeiro. *TTL* é um atributo que define o tempo de validade de um evento. Por exemplo, a localização de um carro pode ser válida por apenas três segundos enquanto que a temperatura de uma sala pode ser válida por meia hora. COPAL irá descartar qualquer evento cuja informação de contexto esteja fora do TTL. Os parâmetros de QoC descrevem a qualidade dos dados/sensores e são opcionais. COPAL oferece suporte aos seguintes: *Source Location*, *Authorization*, *Freshness*, *Trust-worthiness* e *Precision*.

COPAL possui alguns componentes opcionais chamados de (*plugins*). Há um *plugin* específico para a avaliação de QoC: o *QoC Evaluator*. Este plugin dá suporte a apenas três parâmetros e utiliza as métricas desenvolvidas por Manzoor [57] para a avaliação de *freshness*, *trust-worthiness* e *precision*. Além de consultas sobre fluxos de dados contínuos, o COPAL permite também a consulta sobre dados históricos armazenados pelo *plugin* **Context Persistence**. Dados cujos tempos de validade expiram são removidos do histórico. Outro *plugin* oferecido é *Location*, que adiciona a localização de origem do evento. O suporte à distribuição de dados do COPAL

[5]http://esper.codehaus.org

é apenas local. Entretanto os autores do trabalho afirmam planejar uma arquitetura distribuída que possibilite compartilhamento de dados e eventos entre diversos hosts.

5.5 INCOME

INCOME [3,60,61] é um *middleware* de contexto distribuído cujo roteamento dos dados coletados a partir de fontes heterogêneas é baseado no conteúdo e na QoC. O processamento e o consumo de dados de contexto podem ser distribuídos em servidores ou dispositivos móveis com recursos limitados. As funções de processamento (e.g. filtragem, agregação, inferência) sãos definidas utilizando-se JavaScript. O suporte ao desenvolvimento de aplicações do INCOME provê três componentes principais: *Context Collector*, *Context Capsule* e *Broker*. Cada um deles é responsável por uma fase do gerenciamento de contexto. A arquitetura do *middleware* é ilustrada na Figura 5.8.

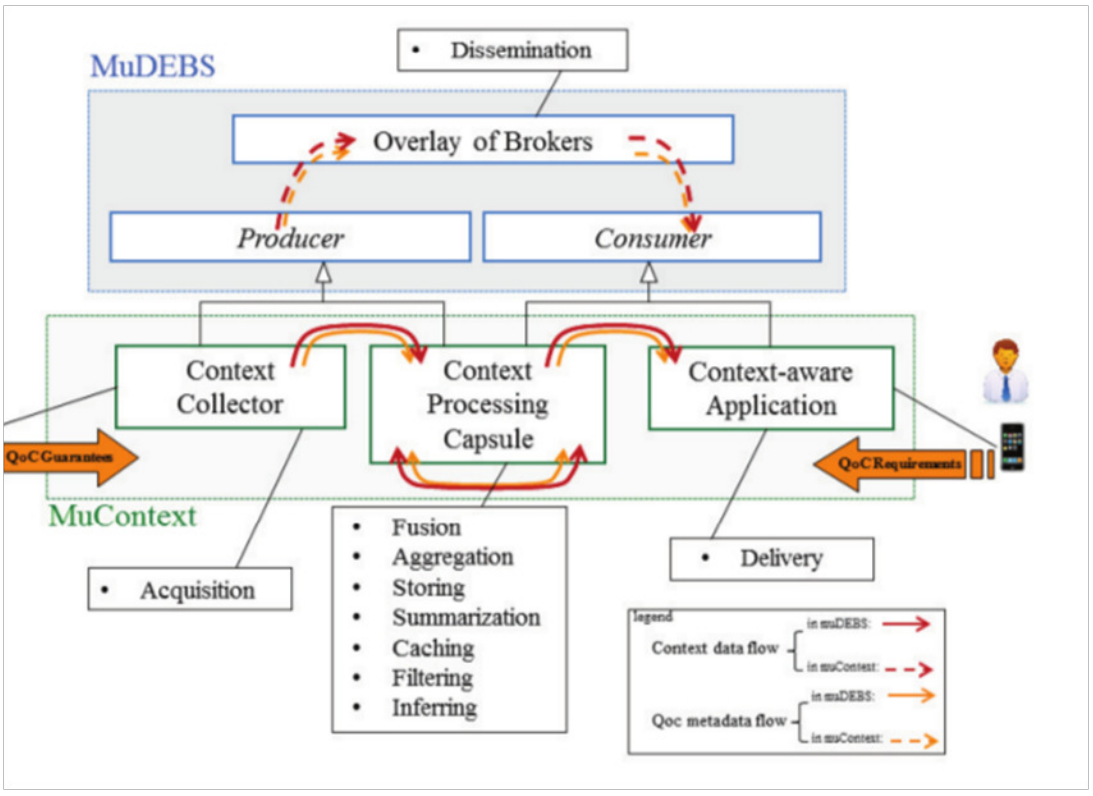

Figura 5.8: Arquitetura do INCOME [61]

O componente *Context Collector* é responsável pela aquisição de dados de baixo nível, ou seja, que ainda não foram processados ou transformados. O INCOME

fornece uma API que permite declarar o tipo de informação de contexto e os tipos de metadados de QoC que os coletores de contexto publicam.

O componente *Context Capsule* é responsável por processar os dados de baixo nível recebidos dos coletores de contexto e transformá-los em informações de contexto de alto nível, que podem ser disponibilizadas para outras cápsulas de contexto ou para as aplicações consumidoras. O *Context Capsule* pode ser tanto um produtor quanto um consumidor e utiliza uma API para especificar suas garantias ou requisitos de QoC respectivamente. Esse componente utiliza os metadados de QoC disponíveis para avaliar a qualidade dos dados como *alta*, *média* ou *baixa*.

Os consumidores de contexto também usam uma API para especificar seus requisitos de contexto e de QoC. As informações de contexto são roteadas até os consumidores apropriados levando em consideração as garantias do produtor e os requisitos do consumidor. A comunicação entre eles é intermediada por *brokers* organizados em uma rede de sobreposição. Cada cliente pode se conectar a um único *broker* por vez. O mecanismo de disseminação do INCOME, baseado em eventos, foi implementado a partir do framework *MuDEBS* (*Distributed Event-Based System*) [52].

Os parâmetros de QoC adotados pelo INCOME refletem a qualidade dos dados/sensores. São eles: *Freshness*, *Precision*, *Completeness*, *Accuracy* e *Spatial Resolution*. Os parâmetros de QoC são incorporados a estrutura do *middleware* seguindo uma abordagem dirigida por modelos[6], a partir de um framework denominado *QoCIM (Quality of Context Information Model)* [62].

5.6 SALES

SALES (*Scalable context-Aware context Aware middleware for mobiLe EnviromentS)* [22, 24, 25, 32] é uma CDDI (*Context Data Distribution Infrastructure*) baseada em QoC para ambientes distribuídos. Nesses ambientes, os nodos que executam uma instância do *middleware* estão organizados logicamente de maneira hierárquica e podem se comunicar para enviar e/ou receber dados de contexto utilizando tanto redes infraestruturadas fixas quanto móveis *ad-hoc*. A arquitetura

[6]O processo de engenharia dirigida por modelos permite possível gerar uma implementação do sistema completa ou parcial a partir do modelo do sistema. [44]

lógica do SALES é ilustrada na Figura 5.9, possui quatro tipos de nodos: CN (*Central Node*), BN (*Base node*), CUN (*Coordinator User Node*) e SUN (*Simple User Node*).

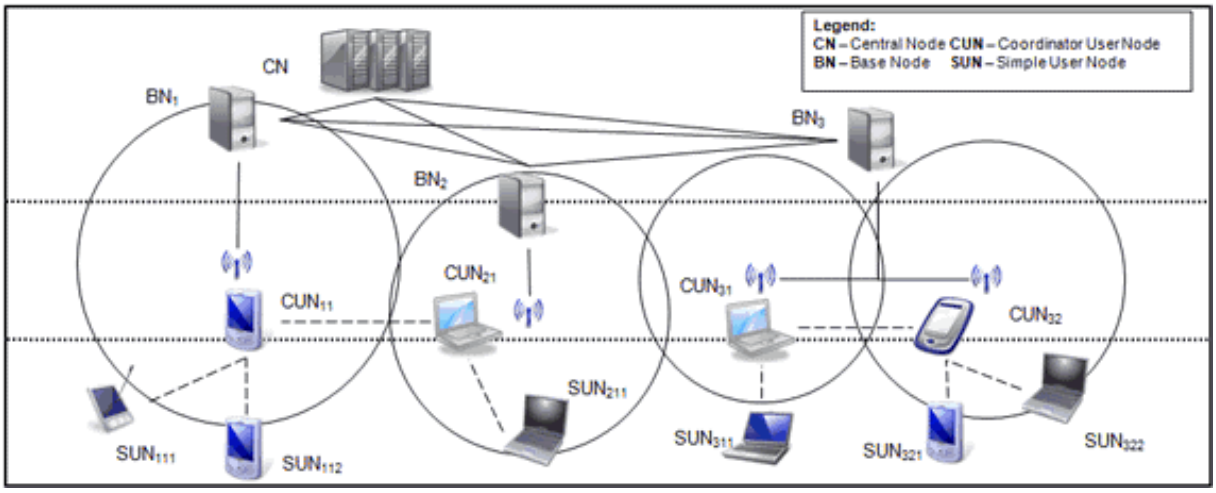

Figura 5.9: Arquitetura Lógica do SALES [22]

O *Central Node* é o nodo raiz que provê serviços de persistência e consulta de dados de contexto. O CN só pode ser acessado diretamente por nodos fixos da mesma LAN. O sistema pode agrupar múltiplos CNs em um cluster a fim de atender requisitos de escalabilidade e disponibilidade. O *Base node* é um nodo fixo que atua como ponto de entrada da infraestrutura fixa. Cada BN gerencia e integra nodos móveis espalhados em diversas redes infraestruturadas móveis (e.g., Wifi, 3G/4G). O *Coordinator User Node* é um nodo móvel especial que agrupa os demais nodos móveis em clusters e se responsabiliza por receber e encaminhar os dados de contexto e consultas oriundas desses nodos para o BN ao qual está conectado via rede infraestruturada e/ou para outros CUNs com os quais estabelece uma comunicação *ad-hoc mult-hop* via Wifi ou Bluetooth. O *Simple User Node* corresponde a cada nodo móvel do usuário que não é o líder do cluster. Cada SUN pode atuar como provedor ou consumidor de contexto. Tal como a comunicação entre CUNs, cada SUN se conecta de modo *ad-hoc* ao seu CUN.

A troca de dados de contexto pode acontecer tanto entre nodos pertencentes a níveis adjacentes como entre nodos pertencentes ao mesmo nível. Inicialmente, o fluxo de dados de contexto é roteado apenas verticalmente da fonte até o CN (chamado de caminho de disseminação de contexto) para que as informações sejam persistidas e ganhem visibilidade. Para orientar a disseminação por diferentes caminhos, o *middleware* divulga **consultas de contexto** que expressam os requisitos de contexto dos consumidores, tanto horizontalmente como verticalmente. Ao receber uma consulta,

o nodo verifica os dados de contexto armazenados localmente e, caso haja uma correspondência positiva, cria uma **resposta de contexto** que será encaminhada de volta para o nodo solicitante utilizando uma solução de roteamento *hop-by-hop*.

O suporte ao desenvolvimento de aplicações do SALES fornece um conjunto de APIs, módulos e componentes pertencentes uma arquitetura divida em duas camadas: Facilidades e Mecanismos. Essa arquitetura é ilustrada na Figura 5.10.

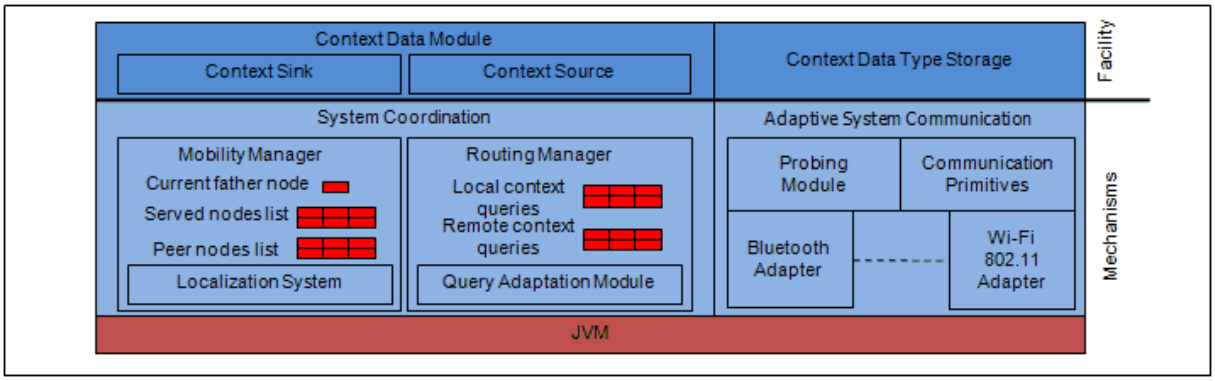

Figura 5.10: Arquitetura de Software do SALES [22]

A Camada de Facilidades implementa funcionalidades de alto nível para envio e recuperação de dados de contexto. O componente *Context Data Type Storage* permite o registro de produtores, consumidores e tipos de dados de contexto por meio de um esquema XML. O *Context Data Module* é um componente genérico que foi especializado em dois sub-componentes: o *Context Source*, que permite criar e publicar dados de contexto no sistema, e o *Context Sink*, que permite recuperar dados de contexto por meio do envio de consultas. Essas consultas fornecem informações de roteamento que serão utilizadas pela camada de mecanismos para a disseminação dos dados de contexto. A Camada de Mecanismos implementa funcionalidades de baixo nível relacionadas ao acesso ao sistema e roteamento de dados de contexto. O módulo *Adaptative System Communication* oferece uma API de comunicação genérica para a troca de mensagens que esconde detalhes específicos da comunicação sem-fio. Funções de comunicação de baixo nível são tratadas pelos componentes *Bluetooth Adapter* e *Wi-Fi 802.11 Adapter*). O módulo *System Coordination* é responsável pelo gerenciamento de mobilidade e seleção da infraestrutura de roteamento. O componente *Mobility Manager* recupera a localização dos nodos e lida com protocolos de gerenciamento de mobilidade cujo o objetivo é manter a organização da estrutura hierárquica em três níveis, garantindo que a cada nodo móvel passa ser concedida uma conexão com um

nodo superior. O *Routing Manager* realiza o roteamento de dados de contexto em cada um dos diferentes níveis da hierarquia. Ele é responsável por receber consultas de outros nodos, de encaminhá-las quando necessário, e de retornar as respostas.

SALES não dá suporte à qualidade associada aos dados e foca na qualidade do serviço de distribuição. Os requisitos QoC são expressos no SALES por meio de um CDDLA (*Context Data Distribution Level Agreement*). SALES considera três parâmetros de QoC. *Freshness*: valor lógico que estabelece limites em relação á idade da informação a ser recuperada; *Data Retrieval Time*: corresponde ao tempo máximo pelo qual o nodo está disposto a esperar por uma resposta de contexto, contado a partir da geração da consulta; *Priority*: especifica a classe do usuário e tem por objetivo permitir a diferenciação de tráfego quando vários dados têm de ser encaminhados. Existem três classes de usuário definidas estaticamente, sendo que os valores dos parâmetros são pré-fixados para cada classe. Classe *Ouro*: tem prioridade 0 e exige receber a versão mais recente do dado em um tempo de recuperação de até 2 segundos; Classe *Prata*: tem prioridade 1 e aceita receber dados válidos (mesmo que não seja o mais recente) em um tempo de recuperação de até 4 segundos; Classe *Bronze*: tem prioridade 2 e aceita receber dados já expirados em um tempo de recuperação de até 6 segundos.

Para que a recuperação de dados ocorra de acordo com o CDDLA, o SALES mapeia automaticamente os parâmetros de QoC para parâmetros de consulta de contexto. Os parâmetros de consulta especificam a forma como a infraestrutura roteará e tratará as consultas. O *Freshness* é mapeado para *Horizontal Time To Live (HTTL)*, que limita o número de saltos da consulta. Por exemplo, se o consumidor requer a versão mais atual do dado de contexto, então o HTTL receberá valor 0, indicando que a consulta deve chegar até o CN. O *Data Retrieval Time* é mapeado para *Query Total Lifetime (QTL)*, que faz com que a infraestrutura descarte a consulta depois que o tempo de validade da mesma expira. Portanto, se o tempo de recuperação exigido é de 2 segundos, então a consulta não pode permanecer válida por tempo maior que esse, já que qualquer resposta encaminhada após esse prazo irá representar tráfego inútil. Por fim, o *Priority* é mapeado para *Query Priority*, que determina a prioridade de encaminhamento do dado conforme a classe.

5.7 Análise dos Trabalhos Relacionados

Marie et. al [60] consideram que a QoC deve estar presente em todas as etapas do gerenciamento de contexto. A análise das abordagens existentes será feita considerando a avaliação de QoC e a capacidade de monitoramento dos serviços com base nos parâmetros de QoC. Os critérios estabelecidos são: suporte à avaliação de QoI e QoS, quantidade de parâmetros de QoC suportados, suporte ao monitoramento de QoC e linguagem de especificação dos requisitos de monitoramento. Para cada critério foi analisado como as diferentes propostas da literatura abordam a qualidade de contexto e/ou aspectos relacionados ao monitoramento. O objetivo é identificar o nível de suporte e as limitações de cada trabalho. A Tabela 5.1 apresenta um quadro comparativo entre os trabalhos relacionados e o M-Hub/CDDL.

	Avaliação de QoC (QoI/QoS)	**Quantidade de Parâmetros de QoC**	**Suporte ao Monitoramento**	**Linguagem de Especificação de Requisitos de QoC**
M-Hub/CDDL	Suporte a QoI e QoS	16	Monitoramento de QoC baseado em CEP abrangendo QoI e QoS	EPL
QoMonitor	Suporte a QoI e QoS (apenas alguns parâmetros)	4	Monitoramento contínuo de QoC	API
AWARENESS	Foca em QoI mas com algum suporte à OoS	4	Monitoramento para descoberta de serviços mas não é dito se atua em QoC	SQL e RDQL
COSMOS	Somente QoI	6	Sem monitoramento	Não possui
COPAL	Bom suporte a QoI e algum suporte a QoS	5	Monitoramento baseado em CEP	EPL
INCOME	Somente QoI	5	Sem monitoramento	API
SALES	Somente QoS	3	Sem monitoramento	CDDLA

Tabela 5.1: Comparativo entre os trabalhos relacionados e o M-Hub/CDDL

5.7.1 Suporte à Avaliação de QoC

Nem todos os dispositivos fornecem informações de qualidade de contexto juntamente com os dados transmitidos. Assim, em alguns casos, o *middleware* precisa computar a QoC de parâmetros não prontamente fornecidos e injetá-la na informação como atributos ou metadados. Isso geralmente é feito durante a fase de processamento (ou pré-processamento). O resultado dessa avaliação pode ser usado como critério de seleção dos provedores. Nesse aspecto, apenas as soluções QoMonitor, INCOME, COSMOS e COPAL utilizam seus respectivos componentes de processamento com

a finalidade de extrair e avaliar a QoC dos dados coletados. Para avaliar a QoC, o QoMonitor utiliza *Assesment Module*, o *middleware* COSMOS usa o componente *QoC Context Node*. O COPAL usa o *plugin QoC Evaluator*. INCOME usa o componente *Context Capsule* para essa mesma finalidade. Entretanto, com exceção do COPAL, os trabalhos referentes a esses sistemas de *middleware* não descrevem quais métricas são utilizada na avaliação de QoC. No trabalho referente ao *middleware* AWARENESS é apenas dito que a fonte de contexto (*Context Source*) informa a sua QoC por meio de metadados, mas não torna explicito a existência de componentes para extrair, processar e avaliar a qualidade de contexto. Nossa proposta possui uma abordagem que avalia tanto QoI quanto QoS. Os parâmetros de QoC podem ser fornecidos pelos provedores mas também podem ser calculados pelo componente *QoC Evaluator*. Além disso, nossa proposta calcula a média dos valores dos parâmetros de QoC e esta informação pode ser utilizada para consulta nos diretórios de serviços. Uma outra característica que difere nossa solução dos demais *middleware* é o uso da avaliação de QoC como mecanismo de apoio à descoberta de serviço. Isso porque o *middleware* disponibiliza as médias das QoC nos serviços de diretórios para ser consultada. Sendo assim, os consumidores podem selecionar os provedores de serviços que melhor atendem seus requisitos de contexto e de QoC.

5.7.2 Quantidade de Parâmetros Suportados

O QoMonitor dá suporte à avaliação de quatro parâmetros de QoC. Entretanto, outros parâmetros podem ser fornecidos pelos serviços como metadados inseridos nas informações de contexto. O AWARENESS não avalia a QoC mas espera que as fontes de contexto especifiquem até quatro parâmetros de QoI (*accucary*, *probability of correctness*, *trustworthiness* e *up-to-dateness*). O COSMOS dá suporte à avaliação de seis parâmetros de QoC (*up-to-dateness*, *trustworhtiness*, *accuracy*, *precision* e *completeness*) mas os autores afirmam que outros parâmetros podem ser facilmente adicionados. O COPAL dá suporte a avaliação de cinco parâmetros de QoC (*source location*, *authorization*, *freshness* e *trustworthiness* e *precision*). O INCOME avalia cinco parâmetros de QoC (*freshness*, *precision*, *completeness*, *accuracy* e *spatial resolution*). Finalmente, o SALES suporta três parâmetros de QoC (*freshness*, *data retrieval time* e *priority*), todos focados em QoS. Nossa proposta possui uma abordagem mais

abrangente pois além de focar tanto em QoI quanto em QoS, avaliar um número maior de parâmetros de QoC que os demais *middleware*. O M-Hub/CDDL avalia seis parâmetros de QoI (*accuracy*, *measurement time*, *available attributes*, *source location*, *measurement interval*, *numerical resolution* e *age*) e nove parâmetros de QoS (*reliability*, *durability*, *history*, *deadline*, *time based filter*, *latency budget*, *lifespan*, *liveness* e *destination order*).

5.7.3 Suporte ao Monitoramento de QoC

O suporte ao monitoramento é uma característica importante pois permite que a aplicação seja notificada quando variações na QoC ocorrem e reagir de acordo. Entre as soluções apresentadas, o QoMonitor conta com um módulo que realiza o monitoramento assíncrono dos serviços com base na QoC. As aplicações podem realizar consultas que ativam um monitoramento contínuo que é responsável por notificar quando as condições da consulta são satisfeitas e chamar um método de *callback* no cliente. AWARENESS fornece um serviço de descoberta de serviço que em seu modo passivo ativa uma espécie de monitoramento para descobrir serviços ativados posteriormente ao momento da consulta, mas não é dito no trabalho se este monitoramento atua sobre parâmetros de QoC. COPAL realiza processamento de eventos CEP com atributos de QoS/QoI em fluxo de dados contínuos, o que configura um monitoramento, além de permitir a consulta de dados históricos. Os demais *middleware* não realizam monitoramento dos serviços com base em informações de QoC. Uma característica que destaca o M-Hub/CDDL, dentre os *middleware* citados, é a capacidade de aplicar o monitoramento de QoC para as consultas contínuas e não apenas ao fluxo de dados de contexto já assinados pelos subscritores. Sendo assim, o consumidor pode monitorar o surgimento de novos provedores de serviços e ser notificado quando algum atender a seus requisitos de QoC. O M-Hub, o QoMonitor e COPAL realizam monitoramento com base em informações de QoC. Apesar disso, o QoMonitor necessita de um outro *middleware* para processar as informações. Já os demais *middleware* não dão suporte ao monitoramento de QoC. O M-Hub/CDDL suporta monitoramento de QoI e QoS e atua sobre mais parâmetros de QoC. Além disso, nosso mecanismo de monitoramento é integrado ao *middleware* de gerenciamento das informações de contexto diferentemente do

QoMonitor. Finalmente, com o M-Hub/CDDL a aplicação pode adicionar e remover regras de monitoramento durante a execução da aplicação sem a necessidade de se reiniciá-la.

5.7.4 Linguagem de especificação de Requisitos de Monitoramento

Dentre as soluções apresentadas, o M-Hub/CDDL e COPAL utilizam CEP para definir os requisitos de contexto. As consultas são especificadas através de EPL. No entanto, COPAL só consegue monitorar fluxos de dados locais enquanto o M-Hub/CDDL monitora fluxos de dados locais e globais (vindos da nuvem CDDL). O QoMonitor e o INCOME utilizam uma API de programação para definir os requisitos de QoC. O SALES utiliza uma estrutura chamada CDDLA para especificar os requisitos de QoC. Finalmente, o AWARENESS suporta consultas utilizando as linguagens SQL e RDQL. O M-Hub utiliza como linguagem de especificação de requisitos para o mecanismo de filtragem e monitoramento a EPL fornecida pela *engine* CEP Asper. A EPL é uma linguagem padrão SQL com extensões próprias que permite às aplicações expressarem requisitos que envolvam correlações, junções, filtragens e agregações a partir de parâmetros de QoC.

6 Conclusões

O desenvolvimento de aplicações cientes de contexto apresenta uma série de desafios devido à natureza dinâmica e heterogênea das fontes de informação de contexto e seus modelos de representação. A heterogeneidade de hardware (sensores, atuadores), protocolos de comunicação, modelos de representação são problemas com os quais as aplicações devem lidar. Além disso, os mecanismos de coleta, processamento, armazenamento e distribuição das informações de contexto apresentam são complexos e representam entraves ao desenvolvimento de aplicações cientes de contexto. Um outro aspecto importante com o qual as aplicações têm que se preocupar é a qualidade das informações de contexto (QoC) que são utilizadas pois, durante o processo de coleta das informações ou durante seu processamento e distribuição, erros podem gerar informações de contexto imprecisas. As aplicações devem poder definir os requisitos de qualidade da informação de contexto que espera receber de modo a cumprir suas funcionalidades.

Ainda em relação à qualidade da informação de contexto, a avaliação dos parâmetros de QoC (informações associadas aos dados de contexto que identificam a qualidade destes dados, como acurácia, resolução numérica dentre outros),a filtragem e o monitoramento da variação destes parâmetros durante a execução de uma aplicação são funcionalidades importantes. A avaliação dos parâmetros de QoC permite que a aplicação conheça o quão uma informação está próxima do seu valor real, a filtragem dos dados permite que o fluxo de informações recebidos possa ser controlado de modo a evitar consumo de processamento e/ou largura de banda desnecessários, enquanto que o monitoramento permite que a aplicação possa reagir a alterações na QoC de forma apropriada. Os desafios inerentes à avaliação dos parâmetros de QoC dizem respeito principalmente à falta de padronização da nomenclatura e à forma de avaliação dos vários parâmetros. Os diversos *middleware* propostos avaliam apenas um pequeno subconjunto dos parâmetros de QoC. Além disso, geralmente tais soluções se concentram em avaliação da Qualidade da Informação (QoI) ou na Qualidade do Serviço de Distribuição(QoS) mas não em ambas. A grade quantidade de dados a serem avaliados também é um problema

que deve-se enfrentar pois o número de sensores e a quantidade de dados que estes geram tende a ser cada vez maior em aplicações de IoT. Já o desenvolvimento de um mecanismo de filtragem e monitoramento apresenta desafios relevantes como a escolha da linguagem a ser utilizada para expressar os requisitos de monitoramento, que deve ser capaz de expressar vários requisitos de monitoramento e ser de fácil utilização, além de ser capaz abstrair detalhes acerca dos diversos tipos de informações de contexto. Outro problema a ser enfrentado é possibilidade de utilização pelo usuário de uma API uniforme para o monitoramento tanto da qualidade da informação (QoI) quanto da qualidade do serviço de distribuição dessa informação (QoS). Também o gerenciamento do impacto do processo de monitoramento no desempenho das aplicações é um desafio, pois o monitoramento pode atuar em uma grande quantidade de dados durante um tempo considerável e deve ser realizado sem afetar o desempenho da aplicação. Finalmente, como os requisitos de monitoramento de informações de contexto das aplicações podem mudar ao longo do tempo, os *middleware* que oferecem tal funcionalidade devem possuir a capacidade de alterações destes requisitos durante a execução das aplicações sem a necessidade das mesmas pararem o processamento.

Diversos *middleware* têm sido propostos para mitigar a complexidade do desenvolvimento de aplicações cientes de contexto, cuja função é esconder dos programadores os detalhes referentes ao gerenciamento do ciclo de vida das informações de contexto, à heterogeneidade de software e hardware, protocolos e outros aspectos, fornecendo ao programador uma API para desenvolvimento de aplicações cientes de contexto em um nível mais alto de abstração. Alguns destes *middleware* fornecem também mecanismos que permitem à aplicação estar ciente da qualidade das informações de contexto.

O *Laboratory for Advanced Collaboration* da Pontifícia Universidade Católica do Rio de Janeiro (LAC-PUC-Rio) em parceria com o Laboratório de Sistemas Distribuídos Inteligentes da Universidade Federal do Maranhão (LSDi-UFMA) desenvolveu um *middleware* voltado ao desenvolvimento de aplicações cientes de contexto associado ao paradigma da IoT denominado M-Hub/CDDL. A utilização do *middleware* permite aos desenvolvedores se concentrar em suas regras de negócio delegando para o M-Hub/CDDL tanto os aspectos de gerenciamento das informações de contexto como da qualidade de contexto associada a essas informações. O

M-Hub/CDDL é baseado em um modelo publicador/subscritor onde provedores de informações de contexto publicam seus serviços e as aplicações interessadas se subscrevem em componentes do *middleware* para receber as informações publicadas.

Este trabalho se concentrou no desenvolvimento de três módulos do M-Hub/CDDL que são responsáveis pelas funcionalidades de avaliação de QoC, filtragem dos dados publicados e/ou recebidos e monitoramento do fluxo de informações de contexto, incluindo a QoC. O objetivo foi oferecer um suporte ao provisionamento de QoC de maneira mais completa abrangendo os aspectos de QoI e QoS lidando com os desafios apresentados.

Inicialmente, apresentamos uma fundamentação teórica sobre ciência de contexto, qualidade de contexto, monitoramento, processamento de eventos complexos e o *middleware* M-Hub. A seguir, fizemos o levantamento do estado da arte referente a *middleware* de contexto para aplicações da Internet das Coisas com algum suporte a provisionamento, monitoramento e adaptação a variações de QoC. No geral, as soluções propostas possuem limitações com relação à avaliação de QoC, seja porque não dá suporte ao mecanismo ou porque fornece apenas um número limitado de parâmetros de QoC ou ainda por não suportar os dois tipos de QoC suportados pelo M-Hub/CDDL (QoI e QoS). As soluções propostas não fornecem mecanismos de filtragem de informações de contexto. Quanto ao monitoramento, apenas três deles fornecem algum suporte limitado à este característica.

Foram apresentadas soluções para cada requisito. Foi desenvolvido um mecanismo de avaliação de QoC que atua tanto na QoI quanto na QoS. Um mecanismo de filtragem das informações de contexto com base em critérios de QoC e nas próprias informações de contexto foi proposto. Tal mecanismo utiliza um Agente de Processamento de Eventos Complexos (EPA) e as aplicações informam seus critérios de filtragem através de uma linguagem de consulta (EPL). Foram projetados mecanismos de monitoramento que atua no fluxo de informações de contexto que visa notificar as aplicações a respeito de variações segundo critérios estabelecidos pelas aplicações. Da mesma forma que o mecanismo de filtragem, o monitoramento utiliza um EPA para processar o fluxo de informações de contexto e os critérios de monitoramento são estabelecidos pelas aplicações através de uma linguagem EPL. Uma das vantagens da utilização de uma EPL de um *engine* CEP para a especificação de critérios de filtragem e monitoramento é que esta é uma linguagem poderosa padrão SQL com extensões

que permite a detecção de eventos em tempo real. Foram realizados experimentos para avaliar os mecanismos de avaliação e monitoramento da qualidade das informações de contexto. Através dos experimentos, constatou-se que os mecanismos se comportam de forma eficiente produzindo os resultados esperados. Os tempos de processamento dos mecanismos, os consumos de memória e bateria foram avaliados e considerados satisfatórios de acordo com os requisitos desejados. A partir destes resultados, pode-se concluir que o desenvolvimento deste trabalho de mestrado ajudou a solucionar alguns problemas enfrentados pelos desenvolvedores de aplicações cientes de contexto no tocante à avaliação e ao monitoramento da qualidade das informações de contexto sendo consumidas.

6.1 Trabalhos Futuros

Ao longo do desenvolvimento do presente trabalho de mestrado, surgiram diversas propostas que podem ser exploradas em trabalhos subsequentes. Dentre estas ressalta-se:

- Apesar do *middleware* atualmente avaliar 15 parâmetros de QoC, consideramos importante tornar a arquitetura mais facilmente extensível para permitir a adição e remoção de novos parâmetros de QoC. Também seria um avanço considerável permitir que a adição e remoção de parâmetros de QoC possa ser feita sem a necessidade de interromper a execução da aplicação.

- Incluir um mecanismo de reconfiguração dinâmica que permita ao *middleware* escolher novos provedores de serviços quando a QoC dos provedores atuais deixar de satisfazer os requisitos de QoC especificados pela aplicação. Atualmente, esta tarefa é de responsabilidade da aplicação.

Referências Bibliográficas

[1] Z. Abid, S. Chabridon, and D. Conan. A framework for quality of context management. In K. Rothermel, D. Fritsch, W. Blochinger, and F. Dürr, editors, *Quality of Context*, volume 5786 of *Lecture Notes in Computer Science*, pages 120–131. Springer Berlin Heidelberg, 2009.

[2] A. Adi, A. Biger, D. Botzer, O. Etzion, and Z. Sommer. Context awareness in amit. In *2003 Autonomic Computing Workshop*, pages 160–166. IEEE Computer Society, June 2003.

[3] J.-P. Arcangeli, A. Bouzeghoub, V. Camps, M.-F. Canut, S. Chabridon, D. Conan, T. Desprats, R. Laborde, E. Lavinal, S. Leriche, H. Maurel, A. Péninou, C. Taconet, and P. Zaraté. INCOME : multi-scale context management for the internet of things. In *AmI '12 : International Joint Conference on Ambient Intelligence*, volume 7683, pages 338–347, Pisa, Italy, Nov 2012. Springer.

[4] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta. *A Survey of Middleware for Internet of Things*, pages 288–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] C. Batista, E. Cavalcate, G. Alves, and P. F. Pires. A metadata monitoring system for ubiquitous computing. In *Proc. of 6th Int. Conf. on MobileUbiquitous Computing, Systems, Services andTechnologies*, pages 60–66, 2012.

[6] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini. A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv.*, 44(4):24:1–24:45, sep 2012.

[7] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180, apr 2010.

[8] C. Bolchini, C. A. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca. And what can context do for data? *Commun. ACM*, 52(11):136–140, nov 2009.

[9] E. Borgia. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54(0):1 – 31, 2014.

[10] N. Brgulja, R. Kusber, K. David, and M. Baumgarten. Measuring the probability of correctness of contextual information in context aware systems. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, pages 246–253, Dec 2009.

[11] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, sep 2006.

[12] R. Bruns, J. Dunkel, S. Lier, and H. Masbruch. Ds-epl: Domain-specific event processing language. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 83–94, New York, NY, USA, 2014. ACM.

[13] R. Bruns, J. Dunkel, S. Lier, and H. Masbruch. Ds-epl: Domain-specific event processing language. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 83–94, New York, NY, USA, 2014. ACM.

[14] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, and J. Lu. Managing quality of context in pervasive computing. In *Quality Software, 2006. QSIC 2006. Sixth International Conference on*, pages 193–200, Oct 2006.

[15] T. Buchholz, A. Küper, and M. Schiffers. Quality of context: What it is and why we need it. In *In Proceedings of the 10th Workshop of the OpenView University Association: (HPOVUA)*, 2003.

[16] D. Cabral Nazario, I. Vilas Boas Tromel, M. Ribeiro Dantas, and J. Leomar Todesco. Toward assessing quality of context parameters in a ubiquitous assisted environment. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6, June 2014.

[17] K. S. Candan, H. Liu, and R. Suvarna. Resource description framework: Metadata and its applications. *SIGKDD Explor. Newsl.*, 3(1):6–19, July 2001.

[18] S. Chabridon, R. Laborde, T. Desprats, A. Oglaza, P. Marie, and S. Marquez. A survey on addressing privacy together with quality of context for context management in the internet of things. *annals of telecommunications - annales des télécommunications*, 69(1-2):47–62, 2014.

[19] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth Computer, Hanover, NH, USA, 2000.

[20] D. Conan, R. Rouvoy, and L. Seinturier. *Scalable Processing of Context Information with COSMOS*, pages 210–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[21] D. J. Cook and S. K. Das. Review: Pervasive computing at scale: Transforming the state of the art. *Pervasive Mob. Comput.*, 8(1):22–35, feb 2012.

[22] A. Corradi, M. Fanelli, and L. Foschini. Implementing a scalable context-aware middleware. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 868–874, July 2009.

[23] A. Corradi, M. Fanelli, and L. Foschini. Adaptive context data distribution with guaranteed quality for mobile environments. In *Wireless Pervasive Computing (ISWPC), 2010 5th IEEE International Symposium on*, pages 373–380, May 2010.

[24] A. Corradi, M. Fanelli, and L. Foschini. Adaptive context data distribution with guaranteed quality for mobile environments. In *IEEE International Symposium on Wireless Pervasive Computing (ISWPC'10)*, pages 373–380, 2010.

[25] M. F. Corradi, Antonio and L. Foschini. towards adaptive and scalable context aware middleware. *Technological Innovations in Adaptive and Dependable Systems: Advancing Models and Concept*, pages 21–37, 2012.

[26] L. Courtrai, F. Guidec, N. Le Sommer, and Y. Mahéo. Resource management for parallel adaptive components. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 134.2–, Washington, DC, USA, 2003. IEEE Computer Society.

[27] R. da Rocha and M. Endler. Foundations of context management in distributed and dynamic environments. In *Context Management for Distributed and Dynamic Context-Aware Computing*, SpringerBriefs in Computer Science, pages 9–27. Springer London, 2012.

[28] A. K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, jan 2001.

[29] J. W. Di Zheng. Research of the qoc based middleware for the service selection in pervasive environment. *IJIEEB*, 3(1):30–37, 2011.

[30] M. Eggum. Smartphone Assisted Complex Event Processing. Masther thesis, University of Oslo, 2014.

[31] C. Emmanouilidis, R.-A. Koutsiamanis, and A. Tasidou. Review: Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. *J. Netw. Comput. Appl.*, 36(1):103–125, Jan. 2013.

[32] M. Fanelli. *Middleware for quality-based context distribution in mobile systems*. PhD thesis, Alma Mater Studiorum Università di Bologna, Maggio 2012. Dottorato di ricerca in Ingegneria elettronica, informatica e delle telecomunicazioni.

[33] J. Filho and N. Agoulmine. A quality-aware approach for resolving context conflicts in context-aware systems. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 229–236, Oct 2011.

[34] J. d. R. M. B. Filho. *CxtBAC: A family of context-based access control models for Pervasive Environments*. Theses, Université Joseph-Fourier - Grenoble I, Oct 2011.

[35] M. Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[36] A. Gaddah and T. Kunz. A survey of middleware paradigms for mobile computing. Technical report, Carleton University, 2003.

[37] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[38] A. M. Gianpaolo Cugola. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3), 2012.

[39] P. D. Gray and D. Salber. Modelling and using sensed context information in the design of interactive applications. In *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, EHCI '01, pages 317–336, London, UK, UK, 2001. Springer-Verlag.

[40] Q. Han, D. Hakkarinen, P. Boonma, and J. Suzuki. Quality-aware sensor data collection. *INTERNATIONAL JOURNAL OF SENSOR NETWORKS*, 7(3), 2010.

[41] A. Held, S. Buchholz, and A. Schill. Modeling of context information for pervasive computing applications. In *Procceding of the World Multiconference on Systemics, Cybernetics and Informatics*. Springer, 2002.

[42] C. Huebscher and A. McCann. An adaptive middleware framework for context-aware applications. *Personal Ubiquitous Comput.*, 10(1):12–20, Dec. 2005.

[43] M. Huebscher and J. McCann. A learning model for trustworthiness of context-awareness services. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 120–124, March 2005.

[44] S. Kent. Model driven engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 286–298, London, UK, UK, 2002. Springer-Verlag.

[45] Y. Kim and K. Lee. A Quality Measurement Method of Context Information in Ubiquitous Environments. In *Hybrid Information Technology, 2006. ICHIT '06. International Conference on*, volume 2, pages 576–581, Nov 2006.

[46] M. Krause and I. Hochstatter. Challenges in modelling and using quality of context (qoc). In T. Magedanz, A. Karmouch, S. Pierre, and I. Venieris, editors, *Mobility Aware Technologies and Applications*, volume 3744 of *Lecture Notes in Computer Science*, pages 324–333. Springer Berlin Heidelberg, 2005.

[47] H. l. Truong, R. Samborski, and T. Fahringer. Towards a framework for monitoring and analyzing qos metrics of grid services. In *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, pages 65–65, Dec 2006.

[48] M. Leclercq, A. E. Ozcan, V. Quema, and J. B. Stefani. Supporting heterogeneous architecture descriptions in an extensible toolset. In *29th International Conference on Software Engineering (ICSE'07)*, pages 209–219, May 2007.

[49] M. Leclercq, V. Quema, and J.-B. Stefani. Dream: A component framework for constructing resource-aware, configurable middleware. *IEEE Distributed Systems Online*, 6(9):1–, sep 2005.

[50] F. Li, S. Sehic, and S. Dustdar. Copal: An adaptive approach to context provisioning. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2010 IEEE 6th International Conference on*, pages 286–293, Oct 2010.

[51] Y. Li and L. Feng. A quality-aware context middleware specification for context-aware computing. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 206–211, July 2009.

[52] L. Lim and D. Conan. Distributed event-based system with multiscoping for multiscalability. In *Proceedings of the 9th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '14, pages 3:1–3:6, New York, NY, USA, 2014. ACM.

[53] I. V. R. V. M. C. F. S. e. S. L.T. Rios, M. Endler. The mobile hub concept: Enabling applications for the internet of mobile things. In *12th IEEE Workshop on Managing Ubiquitous Communications and Services (MUCS 2015), IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March 2015.

[54] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[55] D. C. Luckham and R. Schulte. Event processing glossary - version 1.1. Online Resource. `http://complexevents.com/2008/08/31/event-processing-glossary-version-11/`, July 2008. Last visited: October 2009.

[56] A. Manzoor. *Quality of context in pervasive systems: models, techniques, and applications*. na, 2010.

[57] A. Manzoor, H.-L. Truong, and S. Dustdar. *On the Evaluation of Quality of Context*, pages 140–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[58] A. Manzoor, H.-L. Truong, and S. Dustdar. On the evaluation of quality of context. In *Smart Sensing and Context*, pages 140–153. Springer, 2008.

[59] A. Manzoor, H.-L. Truong, and S. Dustdar. Quality of context: models and applications for context-aware systems in pervasive environments. *The Knowledge Engineering Review*, 29:154–170, 3 2014.

[60] P. Marie, T. Desprats, S. Chabridon, and M. Sibilla. *The QoCIM Framework: Concepts and Tools for Quality of Context Management*, pages 155–172. Springer New York, New York, NY, 2014.

[61] P. Marie, T. Desprats, S. Chabridon, M. Sibilla, and C. Taconet. From ambient sensing to iot-based context computing: An open framework for end to end qoc management. *Sensors*, 15(6):14180, 2015.

[62] P. Marie, L. Lim, A. Manzoor, S. Chabridon, D. Conan, and T. Desprats. Qoc-aware context data distribution in the internet of things. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, M4IOT '14, pages 13–18, New York, NY, USA, 2014. ACM.

[63] D. Marquis, E. Guillaume, and D. Lesenechal. Accuracy (trueness and precision) of cone calorimeter tests with and without a vitiated air enclosure. *Procedia Engineering*, 62:103 – 119, 2013. 9th Asia-Oceania Symposium on Fire Science and Technology.

[64] M. Miraoui, C. Tadj, J. Fattahi, and C. B. Amar. Dynamic context-aware and limited resources-aware service adaptation for pervasive computing. *Adv. Soft. Eng.*, 2011:7:7–7:7, jan 2011.

[65] D. C. Nazario. *CUIDA - Um Modelo de Conhecimento de Qualidade de Contexto Aplicado a Ambientes Ubíquos Internos em Domicílios Assistidos*. Theses, Universidade Federal de Santa Catarina, March 2015.

[66] R. Neisse, M. Wegdam, and M. van Sinderen. Trustworthiness and quality of context information. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 1925–1931, Nov 2008.

[67] G. Pardo-Castellote. Omg data-distribution service: Architectural overview. In *Proceedings of the 2003 IEEE Conference on Military Communications - Volume I*, MILCOM'03, pages 242–247, Washington, DC, USA, 2003. IEEE Computer Society.

[68] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):414–454, First 2014.

[69] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *CoRR*, abs/1305.0982, 2013.

[70] R. M. Pessoa. Infraware: Um middleware de suporte a aplicações sensíveis ao contexto. Master's thesis, Universidade Federal do Espírito Santo (UFES), Vitória, ES, Brazil, 2006.

[71] D. Preuveneers and Y. Berbers. Quality extensions and uncertainty handling for context ontologies. In *Context and Ontologies: Theory Practice and Applications*, pages 62–64, 2006.

[72] D. Preuveneers and Y. Berbers. Architectural backpropagation support for managing ambiguous context in smart environments. In C. Stephanidis, editor, *Universal Access in Human-Computer Interaction. Ambient Interaction*, volume 4555 of *Lecture Notes in Computer Science*, pages 178–187. Springer Berlin Heidelberg, 2007.

[73] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, Sept 1994.

[74] S. Sehic, F. Li, and S. Dustdar. Copal-ml: A macro language for rapid development of context-aware applications in wireless sensor networks. In *Proceedings of the 2Nd Workshop on Software Engineering for Sensor Network Applications*, SESENA '11, pages 1–6, New York, NY, USA, 2011. ACM.

[75] K. Sheikh, D. M. Wegdam, and D. M. van Sinderen. Quality-of-context and its use for protecting privacy in context aware systems. *Journal of software*, 3(3):83–93, March 2008.

[76] K. Sheikh, M. Wegdam, and M. van Sinderen. Middleware support for quality of context in pervasive context-aware systems. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 461–466, March 2007.

[77] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[78] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[79] M. J. van Sinderen, A. T. van Halteren, M. Wegdam, H. B. Meeuwissen, and E. H. Eertink. Supporting context-aware mobile applications: an infrastructure approach. *IEEE Communications Magazine*, 44(9):96–104, Sept 2006.

[80] V. Vieira, P. Tedesco, and A. C. Salgado. Designing context-sensitive systems: An integrated approach. *Expert Systems with Applications*, 38(2):1119–1138, feb 2011. Intelligent Collaboration and Design.