UNIVERSIDADE FEDERAL DO MARANHÃO

Programa de Pós-Graduação em Ciência da Computação

Allinger Lima Medeiros

INavigS: uma infraestrutura de software ciente de contexto para navegação indoor

São Luís
2018

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

Allinger Lima Medeiros

INavigS: uma infraestrutura de software ciente de contexto para navegação indoor

São Luís
2018

Allinger Lima Medeiros

INavigS: uma infraestrutura de software ciente de contexto para navegação indoor

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.

Orientador: Samyr Béliche Vale

Doutor em Ciência da Computação – UFMA

São Luís

2018

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

Medeiros, Allinger Lima.
INavigS : uma infraestrutura de software ciente de contexto para navegação indoor / Allinger Lima Medeiros. - 2018.
99 f.

Orientador(a): Samyr Béliche Vale.
Dissertação (Mestrado) - Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, São Luís, 2018.

1. Computação sensível ao Contexto. 2. Navegação indoor. 3. Serviços baseados em Localização. I. Vale, Samyr Béliche. II. Título.

Allinger Lima Medeiros

INavigS: uma infraestrutura de software ciente de contexto para navegação indoor

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Allinger Lima Medeiros e aprovada pela comissão examinadora.

Aprovada em 27 de Agosto de 2018

BANCA EXAMINADORA

Samyr Béliche Vale (orientador)

Doutor em Ciência da Computação – UFMA

Francisco José da Silva e Silva

Doutor em Ciências da Computação – UFMA

Erico Meneses Leão

Doutor em Engenharia Informática – UFPI

À minha família, meus amigos e meus professores.

Resumo

Computação sensível ao contexto é uma área de pesquisa da computação ubíqua que visa fornecer serviços e/ou informações sem retirar o foco do usuário da tarefa corrente. Serviços baseados em localização, por sua vez, enquadram-se como um tipo especial de aplicações sensíveis ao contexto, onde a localização assume um papel primordial. Devido a evolução das tecnologias de posicionamento e o fato das pessoas passarem a maior parte do tempo no ambiente *indoor*, a demanda por serviços baseados em localização *indoor* não para de crescer. Entretanto, o desenvolvimento dessas aplicações envolve superar diversos desafios. Em particular, para aplicações que fazem o uso de serviços de navegação *indoor*, uma das categorias mais importantes, os desafios incluem: obtenção da localização e orientação do usuário, a representação do espaço *indoor*, a geração de rotas customizadas, a geração de instruções ao longo da rota, o gerenciamento de energia e a inexistência de APIs. Diante desses desafios e objetivando diminuir o esforço necessário para o desenvolvimento dessas aplicações, este trabalho propõe uma infraestrutura de *software* sensível ao contexto e suportada por dispositivos móveis que forneça serviços transparentes de posicionamento e navegação *indoor*.

Palavras-chave: Computação sensível ao Contexto, Serviços baseados em Localização, Navegação *indoor*.

Abstract

Context-aware computing is a research area of ubiquitous computing that aims to provide services and/or information without taking the user's focus away from the current task. In turn, location-based services are categorized as a special type of context-aware applications, where localization assumes a primary role. Because of the evolution of positioning technologies and the fact that people spend most of their time in the indoor environment, the demand for indoor location-based services keeps growing. However, the development of these applications involves overcoming several challenges. In particular, for applications using indoor navigation services, one of the most important categories, the challenges include: obtaining user location and orientation, representing indoor space, generation custom routes, generation of instructions along the route, power management, and non-existence of APIs. In view of these challenges and in order to reduce the effort required for the development of these applications, this work proposes a context-sensitive and mobile-supported software infrastructure that provides transparent indoor positioning and navigation services.

Keywords: Context-aware Computing, Location based services, Indoor navigation.

Agradecimentos

A Deus por tudo.

À minha filha, Maria Rosa Rodrigues Medeiros, por ter suportado, apesar dos inúmeros protestos, os momentos de ausência.

À minha companheira, Michele Simoní dos Anjos Rodrigues, por ter, na medida do possível, me abstraído de questões cuja a minha atenção era necessária.

Aos meus familiares pelo carinho e apoio incodicional que me deram nesta trajetória.

Ao meu orientador, o Prof. Samyr Béliche Vale, pelo apoio, paciência, compreensão e orientação.

Aos professores Erico Meneses Leão e Francisco José da Silva e Silva por terem aceito fazer parte da banca examinadora.

Aos membros do LSDi-UFMA, com os quais compartilhei alegrias, tristezas, e conhecimento, pelos momentos de reflexão, conselhos e estímulos.

Aos membros do Núcleo de Tecnologia da Informação - NTI pelo apoio e por terem possibilitado meu afastamento.

A UFMA e ao PPGCC pela estrutura dada a execução deste trabalho.

"O sucesso é a soma de pequenos esforços repetidos dia após dia."

Robert Collier

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AoA	*Angle of Arrival*.
API	*Application Programing Interface*.
BLE	*Bluetooth Low Energy*.
BR/EDR	*Basic Rate / Enhanced Data Rate*.
CEP	*Complex Event Processing*.
DDS	*Data Distribution Service*.
DI	*Dependency Injection*.
DR	*Dead Reckoning*.
EPL	*Event Processing Language*.
GPS	*Global Positioning System*.
ILBS	*Indoor Location Based Services*.
IMU	*Inertial Measurement Unit*.
INavigS	*Indoor Navigation System*.
ISM	*Industrial, Scientific and Medical*.
KNN	*K-Nearest Neighborg*.
LAC	*Laboratory For Advanced Collaboration*.
LBS	*Location Based Services*.
LSDi	Laboratório de Sistemas Distribuídos Inteligentes.
LST	*Least Squares Trilateration*.

M-Hub *Mobile Hub*.

M-OBJ *Mobile Objects*.

MEPA *Mobile Event Processing Agent*.

MN *Mobile Node*.

MR-UDP *Mobile Realiable UDP*.

PDU *Packet Data Unit*.

POI *Point of Interest*.

PUC-Rio Pontifícia Universidade Católica do Rio de Janeiro.

RFID *Radio Frequency Identification*.

RSS *Received Signal Strength*.

RSSI *Received Signal Strength Indicator*.

RToF *Round-Trip Time of Flight*.

RTT *Round Trip Time*.

S2PA *Short-range Sensing, Presence & Actuation API*.

SDDL *Scalable Data Distribution Layer*.

TDoA *Time Difference of Arrival*.

ToA *Time of Arrival*.

ToF *Time of Flight*.

UFMA Universidade Federal do Maranhão.

URI *Uniform Resource Identifier*.

UUID *Universally Unique Identifier*.

UWB *Ultra-wideband*.

WLAN *Wireless Local Area Network*.

WWDC *Worldwide Developers Conference*.

Sumário

1 Introdução

A popularização de dispositivos móveis, o aumento da importância da Computação Ubíqua, mais precisamente da Computação Sensível ao Contexto, e os avanços das tecnologias baseadas em localização têm levado ao crescente interesse em aplicações e serviços baseados em localização. A Computação Ubíqua se refere a inserção de recursos computacionais no ambiente de tal forma que os mesmos desapareçam de um olhar desatento. De acordo com essa visão, os recursos computacionais devem interagir entre si com o objetivo de prover serviços e/ou informações de forma transparente, sem desviar a atenção do usuário da tarefa em curso [1].

As primeiras demonstrações de Computação Ubíqua foram aplicações sensíveis ao contexto. A transferência de chamadas eletrônicas a partir da atual localização do usuário é um exemplo de funcionalidade implementada em uma dessas aplicações. Computação Sensível ao Contexto é, portanto, uma área de pesquisa da Computação Ubíqua que revoluciona a forma de interação humano-computador, permitindo que serviços e/ou informações sejam fornecidos com o mínimo de intervenção humana [2].

Serviços baseados em localização (*Location Based Services* - LBS) englobam aplicações que dependem da localização do usuário para prover serviços em diversas categorias, incluindo navegação, rastreamento, saúde, educação, segurança e vendas. Em outras palavras, essas aplicações enquadram-se como um tipo especial de aplicações sensíveis ao contexto, onde a localização possui um papel-chave. Espera-se que a construção de LBS para ambientes *indoor* (*Indoor Location Based Services* - ILBS) tenham um grande impacto na sociedade, em virtude das pessoas passarem a maior parte do seu tempo (isto é, 86.9% [3]) nesses ambientes.

Lamentavelmente, nenhuma tecnologia de localização firmou-se como solução única e definitiva para ambientes *indoor* e *outdoor*. No ambiente *outdoor*, a tecnologia mais empregada é o Sistema de Posicionamento Global (*Global Positioning System* - GPS), uma vez que esta possui precisão na casa dos centímetros. Em ambientes

internos, entretanto, o GPS não é adequado pois necessita de linha de visada entre a antena do receptor e os satélites [4].

Ao longo dos últimos anos, diversas tecnologias para a construção de Sistemas de Posicionamento *Indoor* (*Indoor Positioning Systems* - IPS) foram pesquisadas, entre elas: infravermelho, ultrassom, identificação por rádio frequência (*Radio Frequency Identification* - RFID), rede local sem fio (*Wireless Local Area Network* - WLAN), *Bluetooth*, redes de sensores, *Ultra-wideband* (UWB), sinais magnéticos, análise baseada em visão computacional e sons audíveis [4].

Naturalmente, uma solução que funcione satisfatoriamente, em qualquer cenário, não existe. Dessa forma, antes de adotar uma tecnologia para localização *indoor*, é importante considerar diversos parâmetros (por exemplo, custo, acurácia, robustez, escalabilidade, cobertura, etc.), relacionando-os com as necessidades dos usuários [5]. Nesse contexto, o *Bluetooth Low Energy* (BLE) surge como uma alternativa viável, visto que possui baixo custo, é de fácil implantação e é amplamente suportado por dispositivos móveis. A obtenção da posição através de *beacons* BLE vem sendo estudada extensivamente [6] [7] [8] [9] desde a introdução do padrão iBeacon [10], desenvolvido pela *Apple*.

Dentre as categorias de ILBS, a navegação destaca-se como uma das mais relevantes [11], uma vez que é comum pedestres se depararem diante da necessidade de encontrar um determinado destino em espaços desconhecidos. No escopo deste trabalho, a navegação é entendida como a soma entre o cálculo de uma rota viável entre origem e destino e a disponibilização de meios (isto é, exibição da posição do usuário e da rota no mapa ou instruções durante a mesma) que permitam o usuário alcançar tal destino.

O esforço cognitivo e humano empregado na implementação de sistemas sensíveis ao contexto e, consequentemente, de LBS, é considerável. A captura de contexto, por exemplo, quando envolve o uso de sensores, requer conhecimento acerca dos mesmos e a implementação de várias linhas de código. Visando diminuir essa complexidade, diversas ferramentas de apoio ao desenvolvimento de tais aplicações foram desenvolvidas [12] [13] [14] [15]. Todas as soluções propostas, independente da terminologia utilizada, objetivam a separação entre a forma como o contexto é capturado de como ele é efetivamente utilizado. Essa abordagem aumenta a

extensibilidade e a reusabilidade dos sistemas [16] [17]. Uma arquitetura sensível ao contexto, para apoiar o desenvolvimento de novas aplicações, deve fornecer meios para identificar, representar, capturar, inferir ou processar, armazenar, utilizar e avaliar a qualidade de informações contextuais.

Além do problema inerente ao desenvolvimento de aplicações sensíveis ao contexto citado no parágrafo anterior, aplicações que fazem o uso de serviços de navegação *indoor* devem superar alguns desafios, tais como:

- Obtenção da localização e orientação do usuário, visto que existem diversas tecnologias e técnicas que podem ser empregadas, cada uma das quais com problemas intrínsecos e com diferentes níveis de acurácia;
- A representação do espaço *indoor*, já que o modelo espacial deve fornecer meios para extração de rotas e informações simbólicas sobre os espaços;
- A geração de rotas customizadas, em razão da grande demanda por serviços que permitam a personalização das rotas geradas, incluindo preferência pessoais e aspectos relacionados com as restrições de mobilidade de cada usuário (por exemplo, cadeirantes, deficientes visuais, etc.);
- A geração de instruções ao longo da rota, uma vez que as instruções devem ser calculadas e transmitidas ao usuário com o menor atraso possível, evitando desvios desnecessários e, consequentemente, possibilitando que o usuário alcance o destino de forma eficiente;
- O gerenciamento de energia, visto que as aplicações são executadas em dispositivos móveis que, por sua vez, possuem uma fonte de energia limitada;
- A inexistência de *Application Programing Interface*s (APIs), em virtude da disponibilização de APIs ser apontada, por desenvolvedores, como um fator determinante para o sucesso de aplicações ILBS [18].

Diante desses desafios, a disponibilização de uma infraestrutura computacional capaz de munir desenvolvedores com serviços transparentes de posicionamento e navegação *indoor* implicará na redução do esforço e do tempo necessários para o desenvolvimento dessas aplicações. Dessa forma, o desenvolvedor

poderá focar-se na implementação daquilo que trará o real valor para o seu cliente, ou seja, a lógica de negócio da aplicação.

A proposição de uma infraestrutura de *software* que forneça os serviços supracitados é a principal contribuição deste trabalho. Tal infraestrutura recebeu o nome de *Indoor Navigation System* (INavigS). A partir do próprio dispositivo móvel do usuário e de *beacons* BLE distribuídos pelo ambiente, a infraestrutura disponibilizará serviços, tais como: geocodificação, cálculo de rota, provimento de informações sobre o espaço *indoor* e assistência durante a rota. Ao utilizar essa infraestrutura, o desenvolvedor poderá construir uma aplicação que direcione o usuário até uma determinada localização, item ou produto, por exemplo. Isso pode ser útil em diversos domínios: saúde [19], vendas [20] [21], turismo [22], entre outros.

1.1 Objetivos

A pesquisa à qual se refere esta dissertação de mestrado tem como objetivo geral desenvolver uma infraestrutura de *software* ubíqua, sensível ao contexto, baseada em *middleware* e suportada por dispositivos móveis que forneça serviços transparentes de posicionamento e navegação *indoor*.

Os objetivos específicos desta pesquisa são:

- Investigar o estado da arte em relação às arquiteturas sensíveis ao contexto;
- Investigar o estado da arte em relação ao provimento dos serviços de posicionamento e navegação *indoor*;
- Propor uma arquitetura sensível ao contexto que forneça serviços de navegação *indoor*, contemplando a identificação, a captura, a inferência, a utilização e o gerenciamento da informação contextual.
- Projetar e implementar as camadas da arquitetura proposta;

1.2 Estrutura da Dissertação

Esta dissertação está organizada como segue:

- O Capítulo 2 apresenta os principais conceitos, métodos e tecnologias que servem de base fundamental para o desenvolvimento do trabalho.

- O Capítulo 3 descreve a arquitetura proposta, os principais aspectos da implementação e as soluções dadas aos desafios impostos durante o trabalho.

- O Capítulo 4 discute relevantes trabalhos relacionados à navegação *indoor*, descrevendo a arquitetura e as principais funcionalidades de cada um deles. No final, faz-se um análise comparativa entre as infraestruturas encontradas na literatura e a solução proposta por este trabalho.

- O Capítulo 5 apresenta as conclusões obtidas a partir desta pesquisa e apresenta trabalhos futuros que podem ser desenvolvidos a partir deste esforço inicial.

2 Fundamentação Teórica

Neste capítulo é apresentado o arcabouço teórico relacionado à pesquisa desenvolvida por este trabalho. Inicialmente, expõem-se os principais conceitos da Computação Sensível ao Contexto. Aborda-se também aspectos da arquitetura e tópicos relacionados ao processo de desenvolvimento de aplicações sensíveis ao contexto. Em seguida, introduz-se o *middleware Mobile Hub* (M-Hub), com enfoque nos serviços disponibilizados. O M-Hub é uma infraestrutura de *software* executada em dispositivos móveis que atua como *gateway* entre objetos móveis e serviços executados na nuvem, assumindo papel de extrema importância na coleta, pré-processamento e distribuição de dados de contexto. Posteriormente, técnicas para obtenção de posicionamento *indoor* são brevemente descritas, incluindo: Proximidade, Triangulação, Análise de cena e *Dead Reckoning* (DR). Por último, explana-se sobre a tecnologia BLE e sobre o padrão iBeacon.

2.1 Computação Sensível ao Contexto

A história da Computação Sensível ao Contexto iniciou-se a partir da introdução do *Active Badge Location System*, este, considerado por muitos, como o primeiro sistema sensível ao contexto [23] [17]. No entanto, Schilit e Theimer [24], os primeiros a utilizar o termo, o definiram como a capacidade das aplicações de descobrir e reagir às mudanças do ambiente em que elas estão situadas. Para isso, essas aplicações devem constantemente monitorar as informações do mundo que a cercam.

Com o objetivo de ampliar a definição do termo, Abowd et al. [25] analisaram diversos conceitos, subdividindo-os em duas categorias: "usando contexto" ou "se adaptando ao contexto". As definições que se enquadram como "se adaptando ao contexto" requerem que o comportamento da aplicação se modifique para que a mesma seja considerada sensível ao contexto. Já as definições da categoria "usando contexto" continham alguns problemas como: requerer que o sistema sensível

ao contexto capture e interprete o contexto ou que execute serviços automaticamente, limitar a sensibilidade ao ajuste da interface da aplicação e requerer aquisição de contexto em tempo real. Evitando tais limitações, aplicações sensíveis ao contexto foram definidas, por eles, como aplicações que utilizam o contexto para prover relevante informação e/ou serviços, onde a relevância depende da tarefa executada pelo usuário.

2.1.1 Contexto

Inicialmente, o contexto foi definido como a localização, a identidade de pessoas e objetos nas proximidades, bem como, as mudanças nesses objetos [24]. À medida que novas aplicações sensíveis ao contexto foram surgindo, ficou claro que os conceitos, até então utilizados, eram limitados. Definições de contexto baseadas em exemplos dificultam a identificação de novos tipos de informação não citados explicitamente. O uso de sinônimos é outra abordagem bastante comum na definição de contexto, e, da mesma forma que a abordagem anterior, é de difícil aplicação prática [25].

Dessa forma, Abowd et al. [25] definiram o contexto, de forma amplamente aceita, como qualquer informação que possa ser usada para caracterizar a situação de uma entidade (isto é, pessoa, lugar ou objeto) que é considerada relevante para a interação entre o usuário e a aplicação, incluindo ambos.

Categorias

Na tentativa de consolidar o entendimento do que o contexto de fato significa, diversos autores propuseram categorias ou grupos de contexto. Para alguns [25], o contexto primário se divide em quatro categorias: localização, identidade, atividade e tempo. Para outros [17] [26], o contexto se divide em: contexto computacional, contexto do usuário, contexto físico e contexto de tempo. O contexto computacional se refere a informações como conectividade da rede, custos de comunicação, largura de banda, recursos nas proximidades (por exemplo, impressoras, monitores e estações de trabalho), dentre outras. O contexto do usuário agrupa informações como perfil do usuário, localização, pessoas próximas e até mesmo a

situação social. O contexto físico é associado com informações como luminosidade, níveis de ruído, condições de tráfego, temperatura e informações similares. E, por fim, o contexto de tempo se refere a informações como hora do dia, semana, mês e temporada do ano, por exemplo.

Uma abordagem mais recente propõe um contexto unificado, dividido em 6 dimensões (5W1H), que representa a informação contextual centrada no usuário [27]. A dimensão *Who* fornece a identificação do usuário, a *Where* indica a localização do usuário, a *When* representa o tempo em que um contexto está disponível, a *What* está relacionada a informação de um objeto no qual o usuário está prestando atenção, a *How* descreve a expressão de um usuário com sinais, tal como comportamentos (por exemplo, gestos, ação e atividade) ou sinais biológicos (por exemplo, pulso, temperatura e resposta galvânica da pele), e, por último, a dimensão *Why* representa o estado mental do usuário, como por exemplo, a intenção ou a emoção.

2.1.2 Arquitetura de aplicações sensíveis ao contexto

Sistemas sensíveis ao contexto podem ser implementados de diversas maneiras. A abordagem a ser utilizada depende de requisitos especiais e condições, tais como: a localização dos sensores (isto é, local ou remoto), a quantidade de possíveis usuários (isto é, um ou muitos), os recursos disponíveis nos dispositivos utilizados (isto é, PC top de linha ou pequeno dispositivo móvel) ou a facilidade de uma adicional extensão ao sistema. Além disso, o método de aquisição de contexto é muito importante quando projetamos sistemas sensíveis ao contexto porque ele predefine o estilo arquitetural pelo menos em alguma extensão [16]. Existem três categorias para os métodos de aquisição de contexto [28]:

- acesso direto aos sensores: a aplicação coleta o contexto diretamente dos sensores;
- utilização de uma infraestrutura *middleware*: existe separação entre a coleta e o uso do contexto. A infraestrutura esconde detalhes de implementação e comunicação necessários para a aquisição de contexto, possibilitando assim, que o desenvolvimento da aplicação seja focado no uso do contexto;

- utilização de um servidor de contexto: estende a abordagem anterior para permitir o acesso de múltiplos clientes.

Para facilitar a discussão sobre o tema, a Figura 2.1 apresenta uma arquitetura conceitual para aplicações sensíveis ao contexto.

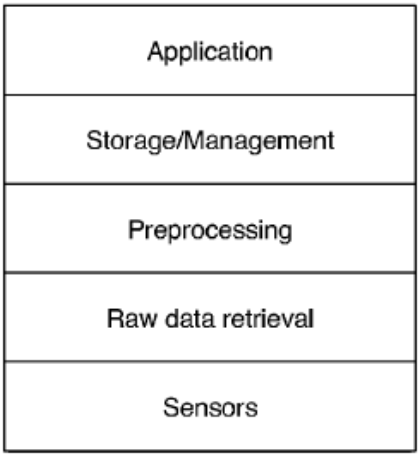

Figura 2.1: Arquitetura conceitual para sistemas sensíveis ao contexto [16].

A camada de sensores se refere a qualquer fonte de dados capaz de fornecer informação contextual útil. A segunda camada é reponsável pela recuperação de dados contextuais não processados. Essa camada esconde detalhes de implementação de baixo nível e, geralmente, é implementada de forma a ser reutilizável. A terceira camada não é implementada em todos os sistemas sensíveis ao contexto e tem como função prover a inferência e a interpretação dos dados contextuais. É possível ainda, nesta camada, realizar a agregação ou composição de contexto e a solução de conflitos entre dados contextuais. A camada de armazenamento e gerenciamento organiza a informação coletada e oferece meios de acesso à informação pela camada superior. A quinta e última camada é a reponsável por prover serviços e/ou informações aos usuários e, portanto, onde a reação a diferentes eventos e instâncias do contexto são implementadas [16].

2.1.3 Desenvolvimento de aplicações sensíveis ao contexto

Diversos desafios ficaram evidentes durante o avanço no desenvolvimento de aplicações sensíveis ao contexto. Um dos maiores desafios encontrados se refere a captura e ao gerenciamento do contexto, pois, como dito anteriormente, o uso de

sensores pode exigir a implementação de várias linhas de código e conhecimento prévio acerca dos mesmos. Portanto, a separação entre a forma como o contexto é capturado de como ele é efetivamente utilizado aumenta a extensibilidade e a reusabilidade dos sistemas [16] [17].

Quanto às questões relacionadas ao desenvolvimento desse tipo de aplicações, Santos [29] propõe um processo que estende as fases de análise e projeto de qualquer ciclo de vida de desenvolvimento de software. Seguindo essa visão, as seguintes fases fariam parte do processo de desenvolvimento: Especificação do Contexto (análise), Projeto do Gerenciamento do Contexto (projeto), Projeto do Uso do Contexto (projeto), Implementação, Teste e Avaliação (manutenção). Dey [30], por outro lado, descreve uma abordagem mais prática e/ou experimental com atividades que, por exemplo, sugerem a instalação de sensores na plataforma necessária e o aprendizado sobre o tipo de informação que os sensores fornecem. Alguns tópicos que devem ser levados em consideração durante as fases de análise e projeto são apresentados nas subseções que seguem.

2.1.4 Identificação

A partir dos requisitos da aplicação, faz-se necessário identificar quais informações contextuais serão necessárias para prover o comportamento desejado. Em sistemas complexos, a identificação das entidades e dos elementos contextuais necessários para caracterizar uma situação pode se tornar uma atividade bastante dispendiosa e, em alguns casos, pode ser até impossível determinar o conjunto de situações, as informações contextuais necessárias para caracterizar uma situação e as ações a serem executadas em uma determinada situação [31].

2.1.5 Representação

Para permitir que a informação contextual seja processada computacionalmente ela deve ser definida e armazenada através do uso de um modelo de contexto [16]. Embora os modelos iniciais abordassem principalmente a modelagem do contexto em relação a uma aplicação ou a uma classe de aplicações, os

modelos genéricos de contexto tornaram-se de maior interesse, uma vez que muitas aplicações podem se beneficiar deles [32].

Um modelo de contexto eficiente deve suportar a derivação de contexto de alto nível a partir de informação contextual não processada e consultas a contexto atual e histórico. Além disso, devem ser fáceis de estender, ou seja, novos tipos de contexto podem ser adicionados sem afetar a informação existente [33].

Existem diversas formas para representar dados contextuais: par chave-valor, linguagens de marcação, modelos gráficos, orientados a objetos, grafos contextuais, baseados em lógica, mapas de tópicos e ontologias.

2.1.6 Captura

Aquisição de contexto se refere ao processo de coleta de contexto a partir várias fontes [33]. No que diz respeito a forma como a informação é adquirida, os sensores foram divididos em três grupos [34]:

- sensores físicos: fornecem a informação a partir do dispositivo físico;
- sensores virtuais: extraem a informação do espaço virtual, ou seja, de aplicações, do sistema operacional, etc;
- sensores lógicos: combinam as informações de sensores físicos e lógicos para inferir informações.

As informações de contexto também podem ser classificadas, considerando a sua fonte e o nível de persistência, como: estática, percebida, proveniente do perfil do usuário ou derivada. As informações estáticas não variam ao longo do tempo e podem ser obtidas de fontes persistentes de dados (por exemplo, tipo de dispositivo e tipo de canal de comunicação). As informações percebidas são altamente dinâmicas e são provenientes de sensores físicos ou lógicos (por exemplo, coordenadas GPS e temperatura). As informações provenientes do perfil do usuário são obtidas a partir do próprio usuário e são consideradas altamente confiáveis em um primeiro momento, mas tendem a caducar com o tempo (por exemplo, nome do usuário, colegas de trabalho, supervisor e permissões). Informações derivadas são obtidas a partir de funções de derivação, que podem variar de uma simples função matemática até um

complexo algoritmo de inteligência artificial (por exemplo, proximidade entre pessoas e dispositivos e localização simbólica) [35] [36].

Como dito anteriormente, a forma que o contexto é capturado predefine o estilo arquitetural em pelo menos uma extensão e, geralmente, a camada de aquisição é implementada de forma a possibilitar o reuso do contexto.

2.1.7 Processamento

Na maioria dos casos, os dados brutos - tal qual são coletados - não possuem valor significativo para aplicações sensíveis ao contexto, portanto, a realização de algum tipo de processamento torna-se essencial. O processamento do contexto pode ser entendido como o conjunto de métodos e processos que realizam raciocínio ou derivação, transformação, combinação e resolução de conflitos de dados contextuais de modo a produzir outras informações mais refinadas, relevantes e coerentes com o que é necessário em um determinado momento. O processamento pode ser dividido em [37]:

- raciocínio: realiza inferência de informação contextual implícita de alto nível a partir de outras informações explícitas de baixo nível;
- transformação: traduz as informações contextuais para um outro formato;
- combinação: combina dados para derivar um novo;
- tratamento de incertezas: trata inconsistências, ambiguidades e incompletude das informações adquiridas automaticamente.

Quanto às técnicas de inferência existentes, existem dois grupos: exatas e inexatas/imprecisas. Técnicas exatas incorporam redes bayesianas, baseados em lógica, em casos, em ontologias e em regras. Por outro lado, as técnicas inexatas incluem as técnicas *fuzzy* e baseadas em evidências [33].

2.1.8 Armazenamento

Alguns sistemas sensíveis ao contexto necessitam de dados históricos para prover informações contextuais mais significativas, assim como estabelecer tendências

e predizer valores de contexto. Alguns *middlewares* armazenam, além dos dados contextuais, regras de inferência e o domínio de conhecimento necessário para a criação de contexto de alto nível [16].

As técnicas de armazenamento de dados desenvolvidas para redes de sensores sem fio não podem ser diretamentes aplicadas para gerenciamento de dados de aplicações sensíveis ao contexto devido as seguintes razões: os dados surgem de múltiplas fontes, os relacionamentos entre diferentes itens de dados precisam ser atualizados e as fontes e consumidores de dados mudam com bastante frequência. Em resumo, os dois problemas que precisam ser observados no armazenamento de dados são: o gerenciamento de grandes volumes de dados e o gerenciamento de relações contextuais entre objetos de dados. Uma possível solução é a utilização de sistemas de armazenamento distribuídos e hierárquicos que, geralmente, suportam armazenamento massivo e podem lidar com as frequentes alterações de dados [33].

2.1.9 Utilização

A utilização é definida como o emprego de elementos de contexto especificados e gerenciados para guiar as variações de comportamento de uma aplicação sensível ao contexto [29]. O uso do contexto pode se dar através dos seguintes serviços [37]:

- assistência às tarefas: visa auxiliar o usuário na execução de suas tarefas e pode ocorrer na forma de recomendações de objetos, materiais ou pessoas;
- percepção: se refere a dar ciência ao usuário sobre o contexto associado a pessoas e interações do seu interesse. Deve-se evitar o provimento de informações de forma intrusiva e que o usuário seja sobrecarregado com informações irrelevantes;
- adaptação: a capacidade do sistema modificar seu comportamento, respondendo de forma oportuna às mudanças ocorridas no ambiente e às ações e definições do usuário;

- outros serviços: o contexto pode ser empregado para enriquecer o conhecimento que já se tem sobre alguma coisa, antes mesmo de se produzir um resultado relevante para o usuário;

Quanto ao comportamento das aplicações sensíveis ao contexto, a execução de ações baseadas na presunção da intenção do usuário podem ocasionar resultados indesejados, uma vez que a inferência de aspectos humanos e sociais, necessários para a tomada de decisão, são extremamente difíceis. Dessa forma, quando necessário, os próprios usuários devem assumir o controle sobre as ações a serem executadas. Para permitir que os mesmos tomem as decisões com base no contexto, foram propostos quatro princípios de projeto que visam atingir a clareza (*intelligibility*) e responsabilidade (*accountability*), são eles [38]:

- informar ao usuário sobre o entendimento e as capacidades atuais do sistema contextual;
- prover *feedback*, incluindo questões como informar sobre a consequência de determinadas ações e solicitar confirmação para ações onde pairam dúvidas;
- forçar a identificação e a publicação de ações, particularmente, quando há o compartilhamento de informações restritas;
- fornecer controle ao usuário sobre o sistema e outras ações que tenham impacto sobre ele, especialmente em casos de conflito de interesses.

2.2 *Mobile Hub* (M-Hub)

O M-Hub [39] é um *middleware* de propósito geral, executado em um dispositivo móvel convencional de uso pessoal, que oportunisticamente descobre, registra e permite a comunicação remota nos modos *unicast*, *group-cast* e *broadcast* para/entre muitos tipos de objetos móveis (*Mobile Objects* - M-OBJs) através do *middleware* de comunicação SDDL. O M-Hub atua, portanto, no espaço localizado entre os serviços disponibilizados no núcleo SDDL/*Data Distribution Service* (DDS) e os M-OBJs presentes em sua vizinhança. Em resumo, os serviços disponibilizados por este

middleware permitem a coleta, o pré-processamento e a distribuição de informações contextuais. Tais serviços são descritos em subseção posterior.

O M-Hub foi projetado e desenvolvido no âmbito do projeto ContextNet[1], por pesquisadores do *Laboratory For Advanced Collaboration* (LAC) da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em conjunto com pesquisadores do Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da Universidade Federal do Maranhão (UFMA).

2.2.1 *Middleware* SDDL

O *Scalable Data Distribution Layer* (SDDL) é um *middleware* de comunicação que conecta nós DDS estacionários, presentes em um núcleo de rede com fio, a nós móveis (*Mobile Node* - MN) a partir de uma conexão de dados sem fio baseada no protocolo IP [40]. Os nós estacionários se dividem em: nós de processamento de dados de contexto e informação; *gateways*, utilizados para comunicação com os MNs; e nós de monitoramento e de controle. Os últimos são capazes de exibir informações contextuais relacionadas aos MNs, enviar mensagens aos mesmos e gerenciar grupos [41]. A Figura 2.2 ilustra a arquitetura do *middleware* SDDL.

Dois protocolos de comunicação são utilizados no SDDL: o *Mobile Realiable UDP* (MR-UDP) e o DDS *Real-Time Publish-Subscribe RTPS* [43] [44]. O MR-UDP é um protocolo usado para a interação entre o núcleo da rede e os MNs. Este protocolo implementa funcionalidades semelhantes às do TCP no topo do UDP e foi personalizado para lidar com conectividade intermitente, travessia *Firewall*/NAT e alterações de endereços IP e interfaces de rede. O DDS é uma especificação do *Object Management Group* (OMG) que descreve um modelo centrado em dados *Publish-Subscribe* (*Data-Centric Publish-Subscribe* - DCPS) para a comunicação e integração de aplicações distribuídas. Esse modelo baseia-se no conceito de um espaço de dados global compartilhado, acessível a todas as aplicações interessadas, e no desacoplamento entre produtores e consumidores de informação.

Alguns dos objetivos que o DDS visa alcançar são: alta performance, eficiente utilização de recursos, escabilidade, flexibilidade, portabilidade e interoperabilidade. O suporte a um conjunto de políticas de Qualidade de Serviço

[1] `http://wiki.lac.inf.puc-rio.br/doku.php`

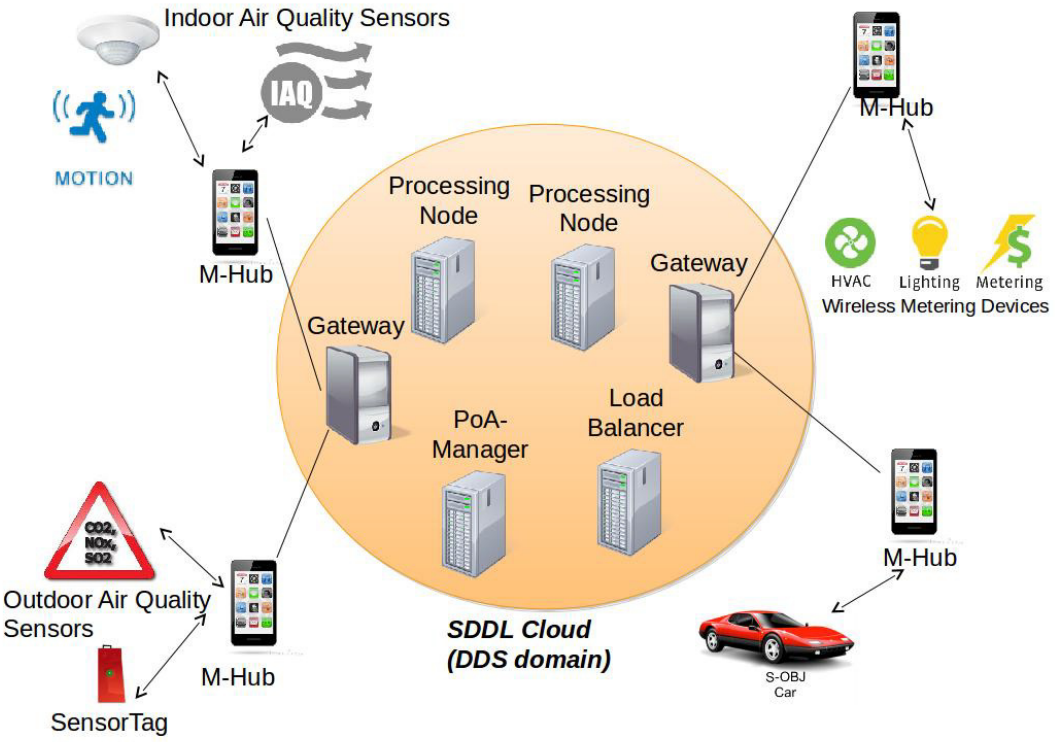

Figura 2.2: Arquitetura do SDDL [42].

(*Quality of Service* - QoS) permite que produtores e consumidores estabeleçam suas capacidades ou necessidades através de contratos que são combinados com o uso do protocolo *Request-versus-Offered*, antes do estabelecimento de qualquer comunicação [45]. Todos os componentes do núcleo SDDL interagem através de tópicos DDS. Tópicos fornecem o ponto básico de conexão entre produtores e consumidores e, no SDDL, alguns são usados para coordenação do próprio *middleware* e outros para a troca de mensagens de aplicação [40].

Como mencionado anteriormente, dentro do núcleo SDDL existem diversos tipos de componentes, cada um dos quais com um papel bem definido. Na borda, localizam-se os componentes chamados de *Gateway*. Estes, funcionam como pontos de ligação (*Point of Attachment* - PoA) e são responsáveis por notificar outros componentes do núcleo quanto a conexão e desconexão de MNs, bem como, por traduzir as mensagens que fluem em ambos os sentidos, ou seja, dos MNs (MR-UDP) para o núcleo da rede (mensagens DDS) e vice-versa.

O *PoA-Manager*, outro componente do núcleo SDDL, é responsável por distribuir periodicamente uma lista de pontos de ligação (*PoA-List*) aos MNs e,

eventualmente, solicitar que alguns destes troquem de *gateway* (*handover*). A ordem da *PoA-List* é relevante, uma vez que se dará prioridade àqueles que estiverem no topo da mesma. Como o *PoA-Manager* é consumidor do tópico DDS onde os *gateways* periodicamente compartilham seus relatórios, ele assume papel fundamental no balanceamento de carga entre os diversos *gateways* disponíveis [46]. Para suportar a entrega confiável durante o *handover*, o serviço opcional *Mobile Temporary Disconnection Service* (MTD), que pode rodar em qualquer nó do núcleo SDDL, coleta todas as mensagens destinadas aos MNs desconectados e as reencaminha tão logo os mesmos voltem a se reconectar. O limite de mensagens armazenadas depende da quantidade de memória alocada para o serviço, uma vez que não há algoritmo para gerenciar a sobrecarga do *buffer* [40].

GroupDefiners são responsáveis por, continuamente, checar e atualizar os grupos aos quais os MNs podem fazer parte. Este componente também localiza-se no núcleo SDDL. Os grupos são especificados pelo desenvolvedor da aplicação, através da implementação do módulo de seleção de grupo (*Group selection module*), e podem ser de longa duração ou definidos a partir do contexto. Para atualizar os grupos definidos por contexto, os *GroupDefiners* se inscrevem em tópicos DDS onde as mensagens ou atualizações de contexto são disseminadas e, de acordo com regras pré-estabelecidas, atribuem a cada nó um ou mais grupos. Em seguida, essa informação é compartilhada com todos os *gateways* através de um tópico DDS de controle. Cada *gateway* possui dois mapeamentos para suportar a gestão dos grupos e comunicação, denominados *MN-to-Groups* e *Group-to-MNs*. O mapeamento *MN-to-Groups* é usado para definir *tags* de grupo para cada mensagem de entrada dos MNs, que posteriomente será verificada pelo *GroupDefiner*. O mapeamento *Group-to-MNs* permite que mensagens sejam enviadas de forma eficiente para todos os MNs de um determinado grupo [41].

Processing Nodes são nós que estão aptos a executar tarefas computacionalmente intensivas, estendendo a capacidade de processamento dos MNs. Cada *Processing Node* pode processar requisições enviadas por um ou mais MNs. Além disso, múltiplos *Processing Nodes*, responsáveis por executar a mesma tarefa, podem ser instanciados.

O *Load Balancer* surge para resolver o problema de divisão de trabalho entre os múltiplos *Processing Nodes* instanciados. A função desse componente é monitorar e readequar, quando necessário, a carga de trabalho executada pelos *Processing Nodes*.

O SDDL utiliza uma solução chamada *Data Processing Slice Load Balancing* (DPSLB), que se baseia no conceito de *Data Processing Slice* (DPS), ou seja, a unidade básica de alocação do fluxo de informação. O balanceamento de carga equivale à redistribuição do número total de DPSs entre os *Processing Nodes*, considerando a atual carga de cada um deles [42].

2.2.2 *Short-range Sensing, Presence & Actuation API* (S2PA)

O S2PA é um protocolo genérico e independente de tecnologia projetado para uniformemente gerenciar a descoberta e a conexão com M-OBJs, utilizando tecnologias de comunicação sem fio de pequeno alcance. Dessa forma, para que M-OBJs de uma tecnologia sejam reconhecido pelo M-Hub é preciso estender uma interface comum (*Technology*), implementando: a descoberta e a conexão com M-OBJs, a descoberta de serviços oferecidos por cada M-OBJ, a leitura e escrita de atributos de serviços e notificações sobre a desconexão de M-OBJs.

2.2.3 Principais Componentes

O M-Hub disponibiliza cinco serviços locais e um gerenciador, todos executando em segundo plano. A arquitetura é ilustrada na Figura 2.3.

O serviço de localização (*LocationService*) é responsável por adicionar a atual localização a cada mensagem enviada para o *Gateway* (GW). Essa localização pode ser estática, manualmente informada ou a última coordenada geográfica obtida pelo sensor GPS.

O serviço S2PA implementa a interface *TechnologyListener* e interage com todos M-OBJs na vizinhança cuja tecnologia WPAN já foi implementada. Esse serviço é responsável por descobrir, monitorar e registrar objetos na proximidade e, dependendo da tecnologia, um *link* de comunicação pode ser estabelecido, sob o qual o M-Hub irá interagir no modo requisição-resposta. Além disso, como cada tipo/marca de M-OBJ possui mensagens com codificação e formato próprio, esse serviço é responsável por traduzir informações e comandos de M-OBJs para objetos Java serializados e vice-versa.

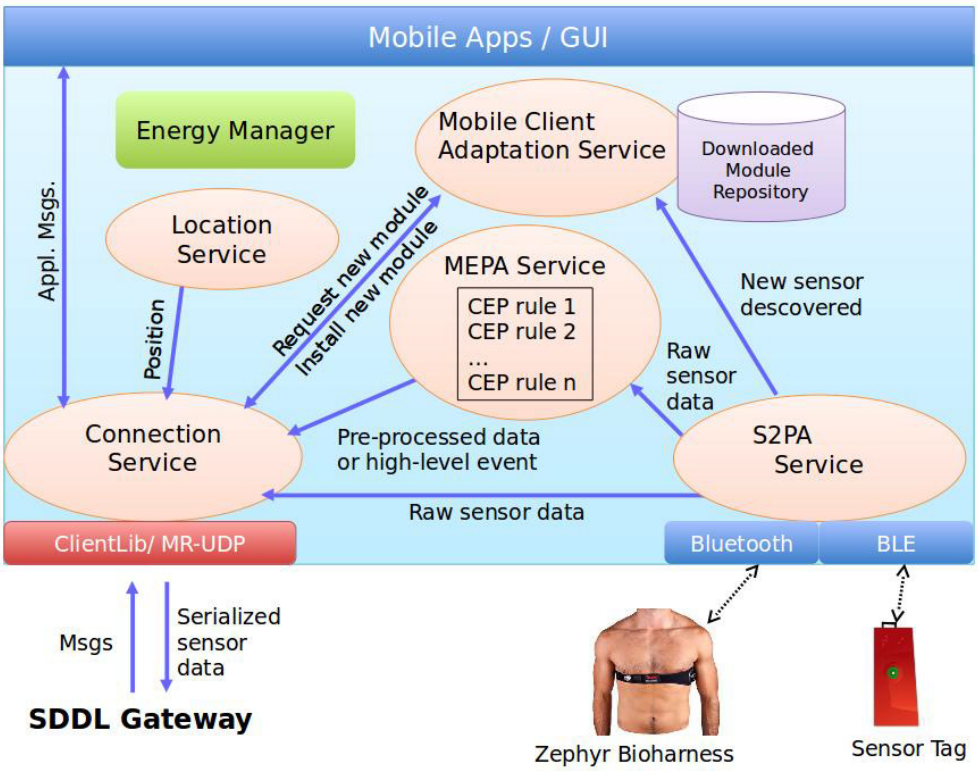

Figura 2.3: Arquitetura do M-Hub [42].

As mensagens são enviadas e recebidas pelo serviço de comunicação (*ConnectionService*) que roda sobre a biblioteca ClientLib. Essa biblioteca implementa a comunicação com o *Gateway* SDDL utilizando o protocolo MR-UDP. Para otimizar o envio de dados, excluindo as mensagens que possuem alta prioridade (mensagens de conexão e desconexão de M-OBJs), informações de diferentes sensores são agrupadas em um simples bloco para transmissão. Esse serviço também implementa um filtro de privacidade que transmite ao núcleo SDDL apenas informações definidas como públicas pelo usuário.

Todos esses serviços têm periodicidade e duração influenciadas pelo atual nível de energia do dispositivo. O gerenciador de energia (*Energy Manager*) é o responsável por esse controle. Em intervalos regulares, ele checa o nível da bateria e se o dispositivo está ou não conectado a uma fonte de energia.

O serviço de adaptação do cliente móvel (*Mobile Client Adaptation Service*) é responsável pela instanciação e gerenciamento do ciclo de vida de código Java recebido dinamicamente. Novos módulos de sensores ou funcionalidades da aplicação podem ser solicitados e recebidos a partir de um serviço que roda no núcleo SDDL,

conhecido como *Adaption Manager*, e carregados em tempo de execução. Localmente, tais módulos são armazenados em um repositório (*Downloaded Module Repository*).

Por fim, para buscar certos padrões no fluxo de dados produzido pelo serviço S2PA, um serviço que utiliza o Asper [47], um motor CEP Esper para Android, foi implementado. Tal serviço recebeu o nome de MEPA (*Mobile Event Processing Agent*). O Asper usa a linguagem declarativa para processamento de eventos *Event Processing Language* (EPL), similar ao SQL, para definir regras CEP. Durante o seu funcionamento, o serviço MEPA escuta todas as mensagens que são enviadas pelo serviço S2PA e, toda vez que um evento é detectado, o envia para o núcleo do SDDL através do serviço de comunicação.

2.3 Métodos de Posicionamento *Indoor*

Como dito anteriormente, diversos sistemas foram desenvolvidos com o objetivo de estudar tecnologias e métodos capazes de resolver o problema de localização em ambientes *indoor*. A escolha do método a ser aplicado deve estar em consonância com a tecnologia de posicionamento selecionada. A combinação entre vários métodos é possível e pode compensar eventuais limitações oriundas de cada um deles. Os métodos podem ser divididos em quatro grupos: Proximidade, Triangulação, Análise de cena e *Dead Reckoning* (DR).

2.3.1 Proximidade

Engloba métodos que são construídos no topo de redes sem fio e usam a informação de conectividade para inferir a posição do objeto alvo, levando em conta os nós de referência em sua vizinhança. Se mais de um nó de referência for detectado, usualmente seleciona-se aquele com a maior intensidade de sinal recebido. A acurácia desses métodos têm relação direta com a densidade de dispositivos espalhados no ambiente e com o alcance do sinal. É de fácil implementação e, geralmente, o custo de implantação é baixo. Dessa forma, para sistemas cuja localização relativa simbólica é suficiente, os métodos desse grupo tornam-se uma boa opção [48].

2.3.2 Triangulação

A triangulação faz uso das propriedades geométricas do triângulo para determinar a localização do objeto alvo [49]. Os métodos se dividem em dois grupos: lateração e angulação (Figura 2.4). O primeiro grupo compreende os métodos que se baseiam na extração da distância, ou seja, métodos baseados no tempo de propagação do sinal e os baseados na intensidade do sinal recebido (*Received Signal Strength* - RSS). O segundo grupo contém o método *Angle of Arrival* (AoA), o qual baseia-se na extração de ângulos.

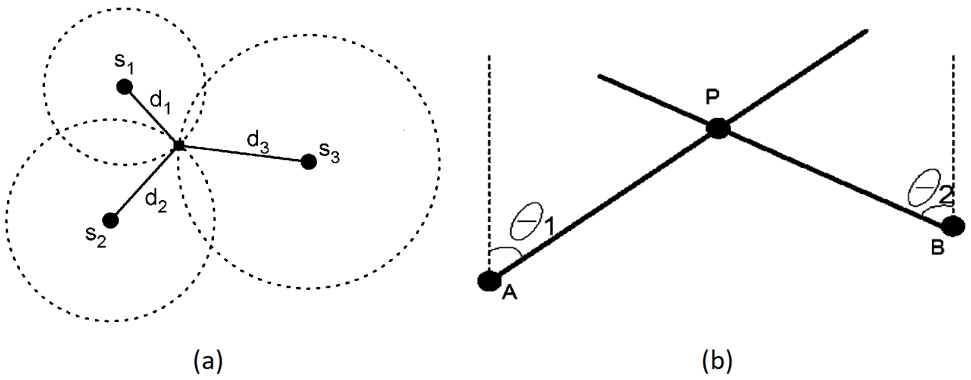

Figura 2.4: (a) Lateração e (b) Angulação. Adaptado de [48] e [50].

Algoritmos de localização baseados em tempo usam o tempo de propagação do sinal entre transmissor e receptor, também conhecido como *Time of Flight* (ToF), para computar a distância entre os mesmos. Esses métodos são suscetíveis a erros produzidos pela imprecisão dos relógios ou erros na estimação do tempo. Apesar disso, esses métodos são, geralmente, muito precisos.

Dentre os métodos baseados em tempo, a abordagem mais simples é conhecida como *Time of Arrival* (ToA). Nessa abordagem, o transmissor inclui o instante da transmissão no próprio sinal transmitido. Em posse desta informação, o receptor é capaz de calcular a distância multiplicando a velocidade de propagação do sinal com o intervalo de tempo decorrido entre transmissão e recepção. Essa abordagem requer que o relógio de todos os envolvidos no processo estejam sincronizados. Para localização no espaço bidimensional são necessários, no mínimo, três diferentes pontos de referência não colineares.

O *Time Difference of Arrival* (TDoA) é outro método baseado em tempo que diminui ou extingue a necessidade de sincronização. Duas implementações são

possíveis. A primeira calcula a diferença de tempo em que o sinal emitido pelo alvo chega a um par de receptores. Uma curva hiperbólica é gerada no espaço de localização para cada par e a localização do transmissor é dada pela interseção de múltiplas curvas, exigindo sincronização de relógios apenas entre os receptores. A segunda abordagem é baseada na diferença de tempo entre dois sinais com tempo de propagação diferentes (por exemplo, ondas eletromagnéticas e acústicas). Nesse método não há necessidade de sincronização.

Outro método baseado em tempo que elimina a necessidade de sincronização é o *Round Trip Time* (RTT) ou *Round-Trip Time of Flight* (RToF). A ideia é medir o tempo necessário para o sinal trafegar do transmissor até o receptor e retornar. Dessa forma, apenas um equipamento é responsável por registrar os tempos de transmissão e chegada do sinal. Para obtenção da posição há a necessidade de se realizar a medição de intervalos para múltiplos dispositivos e, como ocorrem consecutivamente, a acurácia da posição calculada decai para aplicações em que os dispositivo se movem rapidamente, tornando-se esta uma das desvantagens desse método [49].

Métodos baseados em RSS estimam a distância entre o dispositivo alvo e o ponto de referência usando a força do sinal recebido. Esses métodos se baseiam no conceito da atenuação da força do sinal durante a propagação. Modelos teóricos e empíricos são usados para traduzir a diferença entre a força do sinal transmitido e a força do sinal recebido em uma estimativa de alcance. São atrativos devido a alta escabilidade e a sua inerente simplicidade, uma vez que as medidas RSS são nativamente suportadas pela maioria dos *transceivers*. Devido ao severo desvanecimento por multipercurso e sombreamento presente no ambiente *indoor*, modelos de perda de propagação nem sempre se mantém. Os parâmetros empregados nesses modelos devem ser ajustados para cada local [50]. Os métodos são aplicáveis apenas sobre sinais de rádio [49]. A acurácia provida pelos métodos baseados em RSS é negativamente afetada pela distância entre o transmissor e o receptor e, geralmente, é baixa.

O princípio dos métodos de localização baseados em ângulos é similar ao dos métodos baseados em tempo, com a diferença que utilizam ângulos ao invés da distância. Apenas dois pontos de referência são necessários para estimação da posição do dispositivo alvo no espaço bidimensional. O AoA pode ser calculado

através de antenas direcionais ou por um arranjo de antenas. Em ambas abordagens, os receptores são complexos, grandes e caros. A necessidade de *hardware* adicional e a vulnerabilidade ao desvanecimento do sinal (por exemplo, multipercurso e sombreamento) tornam os métodos inadequados para localização *indoor*.

2.3.3 Análise de cena

Os métodos de análise de cena são aqueles que se baseiam na unicidade das características presentes em cada região do ambiente para determinar a localização do objeto alvo. Podem ser empregados em vídeos/imagens ou em sinais eletromagnéticos. Os métodos empregados em sinais eletromagnéticos são compostos por duas fases: fase *offline*, responsável pela captura e extração de características em diferentes posições; e a fase *online*, encarregada de estimar a posição, comparando as características observadas pelo objeto alvo com as informações coletadas anteriormente. O processo de coleta e extração de características costuma ser trabalhoso e demandar tempo, aumentando o custo da implementação. Além disso, ao longo do tempo, a acurácia tende a decair devido às mudanças no ambiente, requerendo a atualização periódica da base de características.

2.3.4 *Dead Reckoning*

Compreende métodos que fazem o uso dos sensores presentes na *Inertial Measurement Unit* (IMU) somados à última posição conhecida para determinar a posição atual. A posição pode ser estimada a partir da dupla integração da aceleração ou adicionando vetores representando a magnitude do passo e a direção do movimento à posição anterior [48]. Como a maioria dos *smartphones* modernos já incluem um IMU, o custo de implementação é baixo. Entretanto, a acumulação, ao longo do trajeto, de pequenos erros oriundos dos sensores é uma desvantagem.

2.4 *Bluetooth Low Energy* (BLE)

O BLE [51] é uma tecnologia sem fio de curto alcance para comunicação entre dispositivos, que tornou-se parte da especificação do *Bluetooth*, na versão 4.0,

adotada em Junho de 2010. O consumo de energia, a complexidade e o custo do *Bluetooth Basic Rate / Enhanced Data Rate (BR/EDR)* são alguns dos problemas atacados pela tecnologia BLE. Apesar das tecnologias BLE e *Bluetooth BR/EDR* não serem compatíveis entre si, ambas podem ser implementadas no mesmo dispositivo.

A faixa de frequência utilizada é a não licenciada *Industrial, Scientific and Medical* (ISM) de 2.4 GHz (2.400 - 2.483). São quarenta canais físicos de 2 MHz, dos quais três são utilizados como canais de anúncio (*advertising channels*) e os demais trinta e sete são usados como canais de dados (*data channels*). Canais de anúncio permitem a descoberta de dispositivos, o estabelecimento de conexões e transmissões *broadcast*. Os canais de dados, por sua vez, permitem a transferência bidirecional de dados entre dois ou mais dispositivos conectados aos pares [52].

Os canais de anúncio 37 (2.402 GHz), 38 (2.426 GHz) e 39 (2.480 GHz) foram alocados em diferentes regiões do espectro para sofrerem menor interferência dos canais IEEE 802.11 [51]. Todos os canais são subdivididos em unidades de tempo conhecidos como eventos. Dispositivos que transmitem pacotes de anúncio são conhecidos como anunciantes (*advertisers*) e aqueles que escutam por tais pacotes sem ter a intenção de conectar-se são chamados de *scanners*. Anunciantes, durante um evento de anúncio, enviam de um a três pacotes nos canais supracitados. Cada pacote de anúncio é enviado em um canal diferente.

Na camada de enlace, os pacotes BLE seguem uma estrutura única, com os seguintes campos: preâmbulo (1 byte), endereço de acesso (4 bytes), o *Packet Data Unit* (PDU) (de 2 a 257 bytes) e o CRC (3 bytes). A diferença entre os pacotes transmitidos nos canais de anúncio e nos canais de dados reside no formato do PDU utilizado. O PDU dos canais de anúncio é composto por um cabeçalho de 2 bytes e uma carga útil (*payload*) de tamanho variado. O tamanho da carga útil deve estar contido entre 6 e 37 bytes e o seu formato depende do tipo de PDU especificado no cabeçalho. Os tipos de PDUs dos canais de anúncio são divididos em três categorias: PDUs de anúncio (*Advertising* PDUs), PDUs de exploração (*Scanning* PDUs) e PDUs de inicialização (*Initiating* PDUs).

Para cada um dos quatros tipos de eventos de anúncio existe um tipo de PDU de anúncio correspondente, a saber: conectável e não direcionado (`ADV_IND`), conectável e direcionado (`ADV_DIRECT_IND`), não conectável e não direcionado

(ADV_NONCONN_IND) e explorável e não direcionado (ADV_SCAN_IND). Todos eles possuem, no mínimo, o endereço MAC do anunciante (6 bytes). Além do endereço MAC, os PDUs ADV_IND, ADV_NONCONN_IND e ADV_SCAN_IND possuem espaço para dados de anúncio cujo tamanho varia entre 0 e 31 bytes. Os dados de anúncio são divididos em estruturas com o seguinte formato: um campo que contém o tamanho da estrutura (1 byte), seguido por um campo que indica o tipo de dado (1 byte) e o dado propriamente dito. Ao invés de dados de anúncio, o PDU ADV_DIRECT_IND possui o endereço MAC do iniciador (6 bytes).

O intervalo entre o início de dois eventos de anúncio consecutivos é a soma de um inteiro entre 20 ms e 10.24 s, múltiplo de 0.625 ms, e um número pseudo-aleatório entre 0 e 10 ms. Para eventos de anúncio explorável e não direcionado e não conectável e não direcionado o múltiplo de 0.625 ms não pode ser menor que 100 ms. O intervalo entre o envio de pacotes deve ser menor ou igual a 10 ms. Para eventos conectáveis e direcionados, no modo ciclo de trabalho elevado, o intervalo entre dois pacotes enviados no mesmo canal deve ser menor ou igual a 3.75 ms. Esse modo de operação deve ser utilizado quando o rápido estabelecimento de conexão for essencial [52].

2.4.1 iBeacon

O padrão iBeacon foi introduzido durante a apresentação *What's New in Core Location* na *Worldwide Developers Conference* (WWDC), ocorrida em junho de 2013. É um padrão proprietário [10], desenvolvido pela *Apple*, que descreve um protocolo de comunicação, construído sobre os eventos de anúncio do BLE, e requisitos de fabricação que os dispositivos, conhecidos como *beacons*, devem atender. Embora o padrão tenha sido desenvolvido pela *Apple*, ele é compatível com qualquer dispositivo que ofereça suporte ao BLE, incluindo aqueles com versão Android 4.3 ou superior [53].

O grande objetivo do padrão é possibilitar a construção de serviços baseados em proximidade e/ou localização. Para tal, um identificador e a intensidade do sinal recebido (*Received Signal Strength Indicator* - RSSI), medido a um metro do *beacon*, são enviados sobre um PDU de anúncio não conectável e não direcionado (ADV_NONCONN_IND). Além disso, o protocolo iBeacon estabelece um prefixo, com 9

bytes e imutável, que engloba cabeçalhos oriundos das estruturas de dados de anúncio, um campo que representa um conjunto de *flags* - obrigatório na versão 4.0 do *Bluetooth* -, o identificador da empresa e o tipo de *beacon*. A figura 2.5 ilustra o pacote enviado.

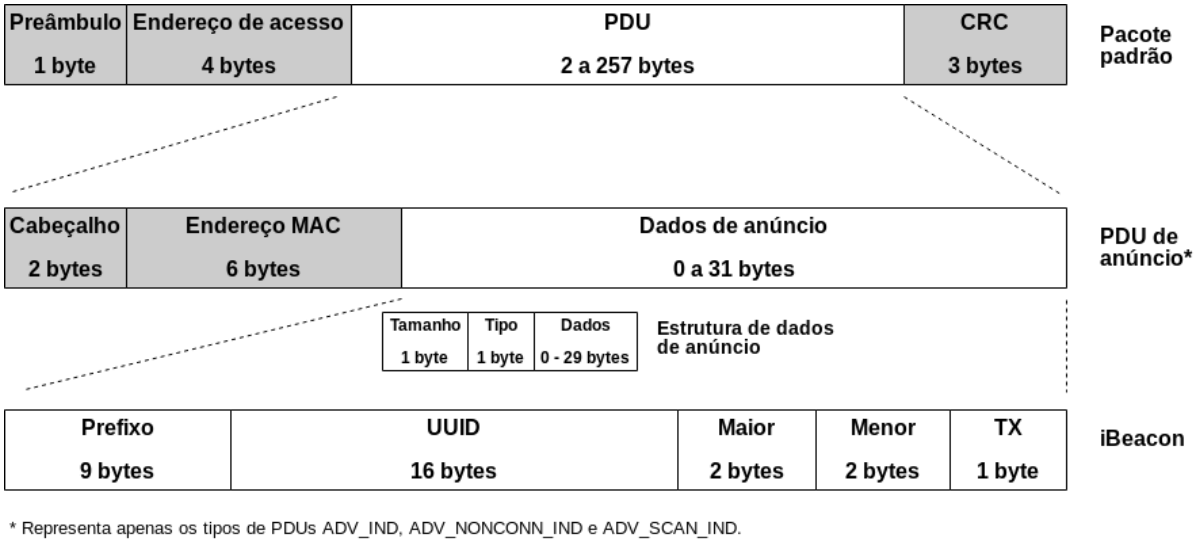

Figura 2.5: Pacote do padrão iBeacon.

O identificador se divide em três partes: *Universally Unique Identifier* (UUID), *Major* e *Minor*. O UUID é um valor de 16 bytes, obrigatório, que identifica unicamente um ou mais *beacons* como sendo de um certo tipo [53] ou pertencendo a uma região [54], empresa [55] ou serviço [56]. O *Major* é um campo opcional de 2 bytes que subdivide os *beacons* com o mesmo UUID. O *Minor*, por sua vez, também é um campo opcional de 2 bytes que identifica unicamente um *beacon* em arranjos de *beacons* com o mesmo UUID e *Major*. O uso e os valores de tais campos devem ser definidos pelo desenvolvedor da aplicação.

A qualidade dos sinais de rádio depedem de características operacionais dos *transceivers* e são afetados por condições ambientais e obstáculos físicos. Reflexão, difração, absorção, refração e dispersão são alguns dos fenômenos que produzem diferentes perdas na intensidade do sinal, mesmo quando a distância entre receptor e emissor são iguais [53]. Dessa forma, o *Received Signal Strength Indicator* (RSSI) enviado serve como valor de referência para auxiliar na definição da distância ou do nível de proximidade entre os dispositivos, quando comparado com o RSSI medido pelo próprio dispositivo receptor. A acurácia também pode ser estimada a partir do RSSI. Quanto mais intenso for o sinal, maior será a acurácia da localização [55]. O

procedimento para o cálculo e a correta aferição do valor a ser transmitido é descrito em [10].

2.5 Considerações Finais

Neste capítulo foram apresentados, resumidamente, os principais conceitos, métodos e tecnologias relacionados com este trabalho. Apesar de terem sido abordadas de forma resumida, o entendimento de tais temas são cruciais para as discussões a seguir.

Apresentou-se os conceitos de contexto, de aplicações sensíveis ao contexto e as principais questões e dificuldades encontradas ao desenvolver tais aplicações. Um modelo conceital de arquitetura para esse tipo de aplicação também foi introduzido. Todas essas informações subsidiam a constatação de que o esforço necessário para a construção de aplicações sensíveis ao contexto pode se tornar um fator impeditivo ao surgimento e a ampla adoção das mesmas. À vista disso, o número de pesquisas que objetivam construir ferramentas para facilitar o desenvolvimento de tais aplicações não para de crescer.

Em relação ao M-Hub, a descrição de suas funcionalidades e características demonstram que esta infraestrutura de *software* pode encurtar o desenvolvimento de aplicações sensíveis ao contexto. As caracterísitcas do *middleware*, inclusive, vão de encontro às necessárias para a construção de aplicações no âmbito da navegação *indoor*, uma vez que é voltado para dispositivos pessoais móveis e permite a descoberta oportunística de sensores baseados em tecnologias WPAN, por exemplo.

A variabilidade e as diferenças dos métodos expostos ilustram a complexidade do problema relacionado à determinação da posição em ambientes *indoor*. As principais lições acerca desse tema são: a estreita relação entre método e tecnologia e a necessidade de avaliação cuidadosa antes do emprego, em cenário real, de um determinado método.

No tocante a seção destinada ao BLE e ao padrão iBeacon, explica-se, resumidamente, algumas características da tecnologia e do padrão adotado por esse trabalho. Em tese, a tecnologia vem sendo considerada promissora para a construção de serviços que fazem uso do posicionamento de pessoas no espaço *indoor*. A

viabilidade é defendida, principalmente, devido ao baixo custo e fácil instalação no ambiente. Apesar disso, é importante ressaltar que não existe, atualmente, tecnologia que se sobressaia em todos os aspectos e, de forma semelhante à escolha do método, deve-se avaliar com cuidado a tecnologia a ser empregada.

3 INavigS

O presente capítulo aborda, em detalhes, a infraestrutura de *software* desenvolvida nesta pesquisa. Apresenta-se, inicialmente, uma visão geral da solução proposta, destacando alguns possíveis cenários de aplicação. Em seguida, aspectos da arquitetura e dos principais elementos que a compõe são discutidos, incluindo particularidades de implementação. Elenca-se, no fim, algumas limitações.

3.1 Visão Geral

O *Indoor Navigation System* (INavigS) é uma infraestrutura de *software* sensível ao contexto e suportada por dispositivos móveis que fornece serviços de navegação *indoor*. Os serviços incluem geocodificação, cálculo de rota, provimento de informações sobre o espaço *indoor* e assistência durante a rota. Facilitar o desenvolvimento de aplicações ubíquas que façam uso de tais serviços é a principal motivação por trás da construção dessa infraestrutura.

O INavigS diferencia-se das outras infraestruturas por: objetivar a fácil implantação em novos espaços, visto que adotou um método de localização por proximidade e um modelo espacial, em tese, simples de ser construído; permitir a customização da rota, mesmo que ainda em pequeno grau; fornecer assistência durante a rota; possuir mecanismos para salvaguardar energia; e, ter disponibilizado API para facilitar a construção de novas aplicações.

Aproveitando-se de dispositivos que implementam a tecnologia BLE espalhados pelo ambiente e do seu próprio dispositivo móvel, o usuário, ao utilizar uma aplicação desenvolvida sobre essa infraestrutura, poderá solicitar instruções para alcançar um destino, por exemplo. Após essa solicitação, a rota será calculada e o usuário passará a receber instruções à medida em que se desloca. Pressupõe-se, para tal, que durante todo o trajeto haverá conectividade com a nuvem ou com o *cluster*, ou seja, com o local onde os serviços estarão sendo executados. Esse é um típico cenário no qual a infraestrutura pode vir a ser utilizada.

A Figura 3.1 ilustra, de forma simples, o uso da infraestrutura de *software* em aplicações de domínios distintos. Uma aplicação no domínio da saúde (*healthcare*), por exemplo, poderia ser voltada ao público de um hospital de urgência e emergência [19]. A comunicação entre equipe médica e paciente, a visualização da localização do paciente por membros da equipe médica e o fornecimento de instruções para guiar o paciente até o local solicitado pela equipe médica poderiam ser algumas das funcionalidades implementadas nesta aplicação. Outra possível aplicação, já no domínio de vendas, poderia ter a função de encontrar produtos em uma determinada loja [20] [21]. E, por fim, no domínio de turismo, uma aplicação focada em guiar e enriquecer a visita de turistas em um museu [22] é um outro exemplo válido.

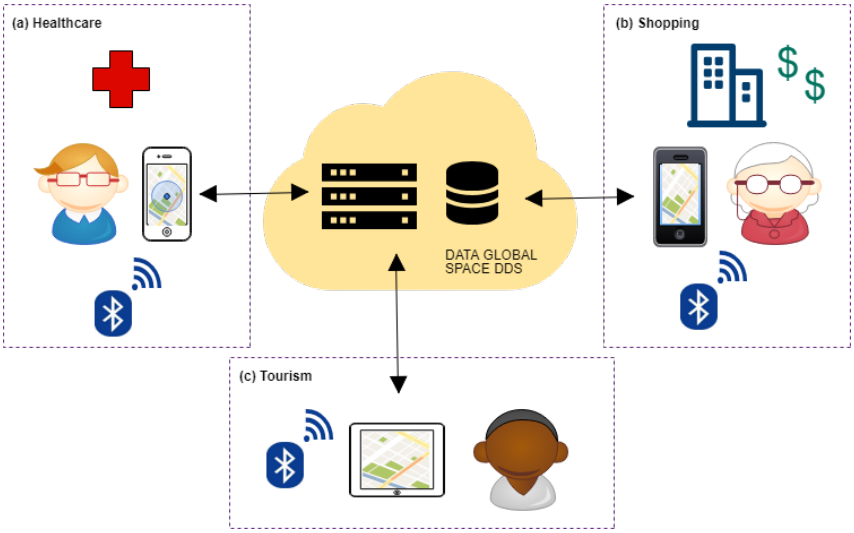

Figura 3.1: Visão geral da arquitetura.

3.2 Arquitetura

A arquitetura do INavigS é distribuída e segmentada em duas camadas, conforme ilustrado na Figura 3.2.

Os serviços de captura, gerenciamento e processamento do contexto são realizados na *Context layer*. O *Context provider* e o *Context manager* são implementados a partir de *softwares* desenvolvidos dentro do escopo de outros projetos e, juntos, provisionam o *Context reasoner* com informação contextual. O componente *Context reasoner*, por sua vez, é produto integral deste trabalho. Os principais serviços oferecidos por esta arquitetura foram implementados neste componente. A troca de

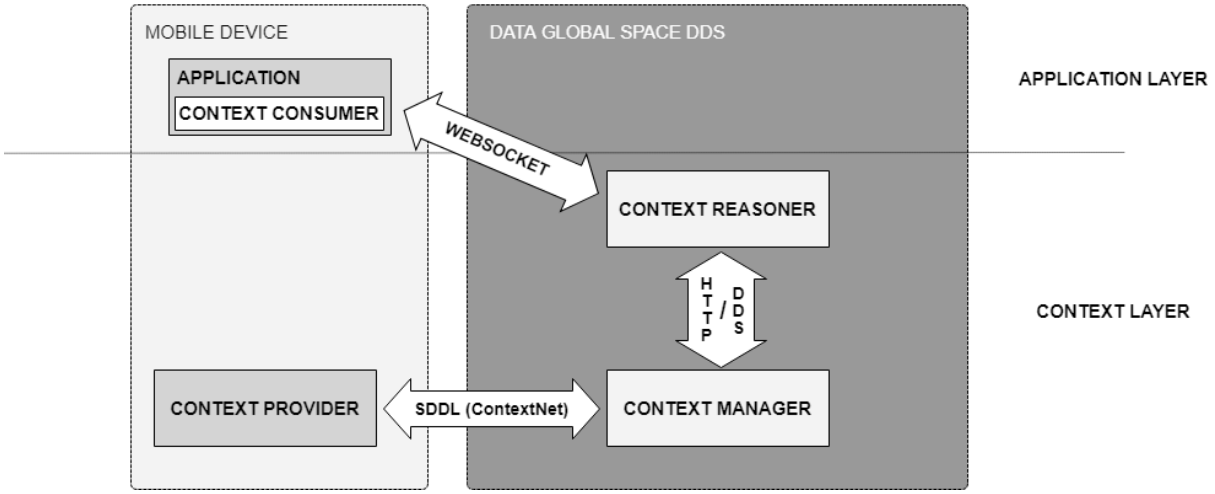

Figura 3.2: Arquitetura.

informações entre o *Context provider* e o *Context manager* ocorre por meio do *middleware* SDDL. Por outro lado, a comunicação entre o *Context manager* e o *Context reasoner* pode ocorrer por meio dos protocolos HTTP ou DDS.

Residem na *Application layer* as aplicações desenvolvidas a partir dessa infraestrutura. Com o objetivo de apoiar a construção de novas aplicações, duas versões de uma biblioteca para o consumo de contexto foram desenvolvidas. Para facilitar a aprendizagem e o uso, as duas versões, apesar de terem sido construídas em linguagens de programação com recursos distintos, mantém abstrações de programação semelhantes. Utiliza-se o protocolo *Websocket* para a comunicação entre a aplicação e o componente *Context reasoner*.

No típico cenário descrito anteriormente, ou seja, de solicitação de assistência durante a rota, uma das formas que os componentes poderiam interagir para atingir sucesso seria: i) durante todo o processo, o *Context provider* enviará ao *Context manager* o último encontro detectado entre o dispositivo móvel do usuário e um *beacon* BLE, em suma, este encontro representará a atual localização do usuário; ii) o *Context manager* armazenará o encontro tão logo ele seja recebido; iii) ao receber a solicitação de assistência durante a rota, o *Context reasoner* irá calcular a rota, levando em conta áreas acessíveis aos deficientes ou não, e iniciará o monitoramento do usuário; iv) será enviada mensagem de confirmação para a aplicação cliente, caso o passo anterior seja realizado com sucesso; v) durante o monitoramento, o *Context reasoner* periodicamente consultará o *Context manager*, para atualização da localização do usuário; vi) caso a localização do usuário mude, o cálculo de uma nova instrução

com base na localização anterior, a atual e o ângulo formado entre o trecho anterior e o próximo é então realizado; vii) em seguida, será enviada mensagem com a nova instrução para a aplicação cliente; viii) os passos v, vi e vii serão repetidos até que a aplicação ou o usuário solicite o cancelamento. O diagrama de sequência que ilustra a interação descrita acima é retratado no Apêndice A - Figura A.2. Nas próximas seções os principais elementos dessa arquitetura serão descritos em detalhes.

3.2.1 *Context Provider*

O componente *Context provider*, pertencente a *Context layer*, é responsável por capturar, representar e realizar o pré-processamento do contexto. A sua implementação faz uso do *middleware* M-Hub e, dentre os diversos serviços oferecidos por este *middleware*, três foram utilizados: *S2PA Service*, *MEPA Service* e *Connection Service*.

No que tange este trabalho, a informação contextual de interesse é a localização do usuário no espaço *indoor*, representada como um encontro entre o dispositivo móvel do usuário e um *beacon* BLE. Portanto, o método de posicionamento utilizado é por Proximidade, ou seja, considera-se como a posição do dispositivo móvel a posição do *beacon* com maior RSSI medido. Os *beacons* implementam o protocolo iBeacon, descrito em seção anterior.

Durante a fase de escaneamento da tecnologia BLE, o serviço *S2PA Service* cria eventos que registram a descoberta de dispositivos. Cada evento é representado através da classe java `SensorData` que possui dentre outras informações, as seguintes: identificador do M-OBJ, no caso de *beacons* é o próprio UUID, e o RSSI. A extração do UUID dos pacotes de anúncio do BLE foi realizada com o auxílio da biblioteca Android Beacon Library[1], versão 2.12.2. Essa biblioteca é oriunda de um projeto *open source* mantido pela empresa Radius Networks[2].

Além dos eventos de descoberta, existem os eventos que se originam no início e no fim de cada período de escaneamento. No intuito de economizar energia, os períodos de escaneamento são separados por intervalos de tempo que variam de acordo com o nível de bateria medido pelo componente *Energy Manager*.

[1] `https://altbeacon.github.io/android-beacon-library/`

[2] `https://www.radiusnetworks.com/`

Atualmente, três configurações são possíveis (*LOW*, *MEDIUM* e *HIGH*). Dessa forma, tais eventos são utilizados para demarcar o intervalo no qual os eventos de descoberta devem ser conjuntamente analisados. Esses eventos são representado pela classe `TechnologyData`, que contém o identificador da tecnologia e a ação realizada (`TechnologyData.START_SCAN` ou `TechnologyData.STOP_SCAN`). Todos os eventos do serviço S2PA são enviados para processamento no serviço MEPA.

No serviço MEPA, três regras CEP foram adicionadas com o objetivo de avaliar qual *beacon* encontra-se mais próximo do dispositivo móvel. A primeira regra (Apêndice B - Listagem B.1) utiliza os eventos que demarcam o início e o fim do período de escaneamento para a criação de um contexto dentro do motor CEP. Contexto, em EPL Esper[3] [57], é uma forma de classificação de eventos em um ou mais conjuntos, sob os quais consultas EPL poderão atuar de forma independente dos demais. Os conjuntos são chamados de *context partitions*. Um termo alternativo para *context partition* é *event processing agent instance*.

A segunda regra (Apêndice B - Listagem B.2) insere no fluxo de eventos `BeaconData` o evento associado ao *beacon* com maior RSSI medido, levando em consideração o contexto criado pela regra anterior. A terceira regra (Apêndice B - Listagem B.3) consulta, a cada segundo, o último evento do fluxo de eventos `BeaconData`. O resultado é enviado para a nuvem através do serviço *Connection Service*. Até que um novo período de escaneamento seja completamente processado, essa regra considerará o último encontro registrado como válido.

De forma resumida, o diagrama de sequência ilustrado na Figura A.1 do Apêndice A demonstra a interação entre os subcomponentes do *Context Provider* durante o processo de descoberta. Após a inicialização das tecnologias implementadas (passo 1), no caso concreto a tecnologia BLE (`BLETechnology`), o serviço S2PA passa a desencadear periodicamente o processo de descoberta de cada tecnologia. No início de cada fase de busca, o S2PA gera um evento, o envia para o serviço MEPA e inicia a busca propriamente dita, enviando mensagem para `BLETechnology` (passos 2 e 3). Como descrito anteriormente, ao receber esse evento, o motor CEP, presente no serviço MEPA, cria um contexto para avaliação dos eventos de descoberta. Ao encontrar M-OBJs durante o período de busca, a classe `BLETechnology` notifica o

[3] `http://www.espertech.com/esper/`

`TechnologyListener`, que por sua vez, extrai as informações necessárias, cria um evento de descoberta e o envia também para o serviço MEPA (passos 4 e 4.1). No final do processo, o serviço S2PA envia mensagem para `BLETechnology` terminar a fase de busca e notifica o serviço MEPA do término da mesma, em seguida, com evento próprio para esse fim (passos 5 e 6). A partir desse momento, o motor CEP encerra o contexto, avalia o evento de descoberta com maior RSSI e passa a enviá-lo para a nuvem, periodicamente (passo 7).

3.2.2 *Context Manager*

O armazenamento das informações contextuais e o processamento de consultas relacionadas são de responsabilidade do *Context Manager*. Este componente pertence a *Context layer* e é implementado com o uso do HORYS[4], *software* escrito em linguagem Java e desenvolvido dentro do escopo do projeto Hospital 4.0 da PUC-Rio. Em poucas palavras, o HORYS registra encontros entre dispositivos móveis, executando o M-Hub, e objetos móveis (M-OBJs). Os encontros são armazenados em banco de dados NoSQL. Na esfera do INavigS, os M-OBJs são *beacons* BLE. Para realização das consultas, o paradigma de comunicação adotado é o requisição-resposta (*request-response*).

Internamente, possui um componente que centraliza o processamento de consultas, independente se as mesmas chegam através de requisições sobre o DDS ou via serviços web (HTTP). Todas as requisições endereçadas a este componente devem seguir um protocolo, conhecido como HORYS API *Protocol* (Apêndice B - Listagem B.4). A mensagem do protocolo é composta pelos seguintes campos: *mode*, que indica se a mensagem se refere a uma requisição ou a uma resposta; *operation*, refere-se a operação solicitada; e *parameters*, um conjunto de pares chave-valor para transferência de informações adicionais.

Além do serviço *Put Rendezvous*, cuja função é registrar os eventos de encontro enviados via DDS, três consultas são disponibilizadas, a saber: *Connected Things*, que retorna todos os M-OBJs conectados a um M-Hub; *Connected M-Hubs*, responsável por fornecer todos os M-Hubs conectados a um M-OBJ; e, o *AVG Rendezvous Duration* que retorna o tempo médio em que cada M-Hub

[4] `http://interscity.org/software/contextnet/`

permanece conectado a um determinado M-OBJ, além do tempo médio entre todos esses encontros. Todos os serviços e consultas sofrem a influência do parâmetro `updateperiod`, configurável através de arquivo de propriedades, que mantém o tempo máximo entre dois eventos de encontro consecutivos e, na prática, é o tempo utilizado pelo HORYS para detectar uma desconexão.

As listagens B.5 e B.6 do Apêndice B se referem às respostas das consultas *Connected M-Hubs* e *Connected Things*, respectivamente. Em ambas estão presente os campos *response*, que contém um conjunto de UUIDs em resposta à operação solicitada, e *operation*, indicando a consulta realizada. Os campos *thingID* e *mhubID* representam os parâmetros de entrada e são, respectivamente, o UUID do M-OBJ e o UUID do dispositivo móvel.

3.2.3 *Context reasoner*

Como o próprio nome sugere, o componente *Context reasoner* é responsável por realizar a inferência sobre o contexto, produzindo informações com maior valor agregado. Este componente foi desenvolvido completamente no âmbito deste trabalho e, da mesma forma que os componentes anteriores, localiza-se na camada *Context layer*. A conversão de *beacons* em espaços semânticos e o fornecimento de rotas e instruções para o cumprimento das mesmas são alguns dos serviços oferecidos por este componente. Os serviços são disponibilizados por meio do protocolo *Websocket*.

Modelo espacial

Para apoiar a inferência, utiliza-se um modelo espacial simbólico hierárquico. O espaço foi dividido em três níveis: prédio (`Building`), elementos do prédio (`BuildingElement`) e sensores (`Beacon` e `Link`). Informações como o nome do prédio e o endereço residem no nível prédio. Elementos do prédio descrevem unidades simbólicas elementares e podem ser do tipo entrada (por exemplo, entrada principal e saídas de emergência), sala, banheiros ou caminho (por exemplo, *hall*, corredores e escadas). Além disso, os elementos podem ser verticais, no caso de possibilitarem a mudança de nível (por exemplo, escada), e acessíveis, ou seja, quando cadeirantes podem acessá-lo ou atravessá-lo. O nível de sensores incluem

os nós e as arestas, que demonstram a existência de conectividade entre os nós. Cada nó possui associado a si o UUID e a coordenada geográfica do *beacon* ao qual representam. As arestas contém o peso, a distância entre os nós em metros, utilizado para obtenção do menor caminho durante a execução do algoritmo Dijkstra. Em relação à implementação, o modelo é armazenado em forma de tabelas em uma base de dados relacional PostgreSQL[5].

Uma possível abordagem prática para a construção do modelo espacial (Figura 3.3 (a)), a partir do nível de sensores, consiste em: i) obter as coordenadas geográficas de uma posição base (por exemplo, entrada principal), representada pelo ponto P0; ii) obter o azimute (isto é, α ou β) e a distância (isto é, d1 ou d2) para cada um dos pontos onde os *beacons* estarão posicionados (isto é, P1 ou P2), tendo como base o ponto imediatamente anterior. Diante dessas informações, torna-se possível calcular as coordenadas de todos os pontos mapeados. É importante notar que essa abordagem não necessita de modelo espacial anterior (por exemplo, desenho CAD, planta baixa, etc.), facilitando a adoção em espaços com pouca informação disponível.

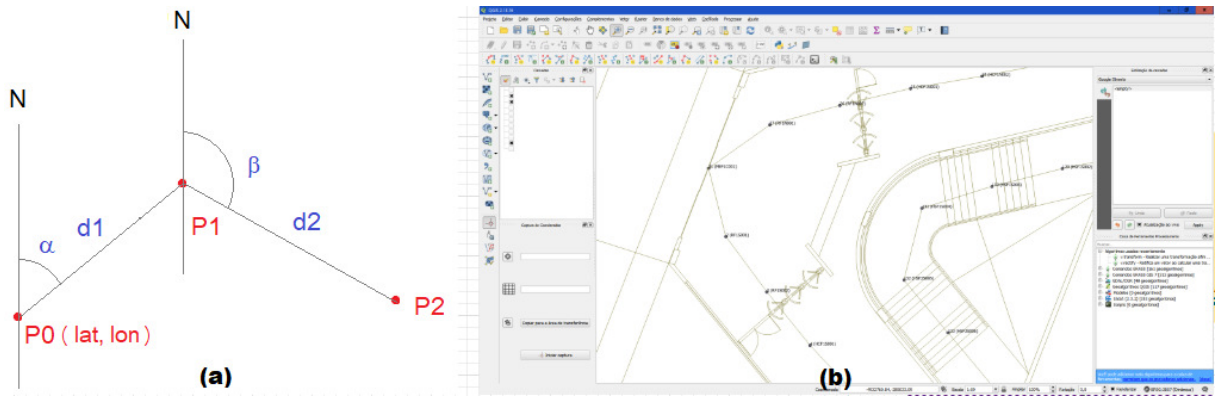

Figura 3.3: Abordagens para a construção do modelo espacial - (a) prática e (b) pragmática.

Caso seja possível obter uma camada contendo dados geoespaciais em forma de vetor (por exemplo, *shapefile*) representando o espaço *indoor*, pode-se adotar uma abordagem mais pragmática (Figura 3.3 (b)). Utilizando uma aplicação como o QGIS[6], pontos de referência podem ser previamente definidos. A obtenção das coordenadas de cada ponto de referência torna-se tarefa trivial (*plugin* Captura de coordenadas). A instalação dos *beacons* nos pontos de referência, provavelmente, será efetuada de forma semelhante a abordagem prática.

[5] https://www.postgresql.org/
[6] https://qgis.org/en/site/

Principais subcomponentes

Internamente, seis subcomponentes foram desenvolvidos, a saber: `GeocodingService`, responsável pela tradução de *beacons* em coordenadas ou espaços semânticos e vice-versa; `BuildingInformationService`, encarregado de prover informações sobre o espaço interno; `RoutingService`, subcomponente que fornece rotas; `GuidanceService`, cuja função é prover assistência durante a rota; `MonitoringService`, incumbido de monitorar o deslocamento dos usuários; e, `ProtocolTranslatorService`, responsável por resolver as requisições realizadas via *Websocket*.

Como mencionado anteriormente, o subcomponente `GeocodingService` é encarregado da função de tradução de *beacons* em coordenadas ou espaços semânticos e vice-versa. A partir das consultas implementadas é possível obter a localização e a coordenada geográfica (`BuildingElement` e `Coordinate`) de um determinado *beacon* (`getLocationByBeacon` e `getCoordinateByBeacon`) ou da atual posição do usuário (`getLocationByCurrentPosition` e `getCoordinateByCurrentPosition`). Também é possível traduzir uma localização em um *beacon* (`getBeaconByLocation`). A atual posição do usuário é consultada no *Context manager*.

O `BuildingInformationService` fornece informações relacionadas ao prédio em si e aos elementos que o compõem. Atualmente, dois tipos de consulta estão disponíveis. O primeiro tipo permite buscar informações sobre um prédio a partir de um determinado *beacon* (`getBuildingByBeacon`), do identificador do prédio (`getBuildingById`) ou da atual posição do usuário (`getBuildingByCurrentPosition`). O segundo tipo busca por uma coleção de lugares (`BuildingElement`) pertencentes a um determinado prédio. Esse tipo de consulta fornece meios para popular lista com possíveis destinos. De forma análoga a anterior, três opções são disponibilizadas: `getPlacesByBeacon`, `getPlacesByIdBuilding` e `getPlacesByCurrentPosition`.

O `RoutingService` permite que rotas, incluindo as acessíveis, sejam consultadas. De forma semelhante aos serviços supracitados, é possível obter rotas a partir da atual posição do usuário sem que a mesma seja expressamente informada.

A atual implemetação faz uso da biblioteca Java JGraphT[7]. Esta, fornece algoritmos e objetos baseados na teoria de grafos. Primeiramente, constrói-se um grafo ponderado não direcionado (`SimpleWeightedGraph`) a partir das informações contidas no nível de sensores do modelo espacial (`Beacon` e `Link`). Se a consulta levar em conta apenas espaços acessíveis, os *beacons* associados a lugares (`BuildingElement`) marcados como inacessíveis serão desconsiderados. Atualmente, essa é a única customização disponível. Após a construção do grafo, roda-se o algoritmo Dijkstra (`DijkstraShortestPath`) para obtenção do menor caminho.

O `MonitoringService` é responsável por, continuamente, interagir com o *Context manager*, mais precisamente através da consulta *Connected Things*, para rastrear alterações no posicionamento de usuários de interesse. Ao executar o método `createMonitor(UUID clientID)`, uma *thread* é criada e escalonada (`ScheduledThreadPoolExecutor`) com atraso fixo (`scheduleWithFixedDelay`) de um segundo. Em outras palavras, em intervalos regulares, a *thread* realizará consulta ao *Context manager* e, em caso de mudança no posicionamento do referido usuário, a implementação do *listener* `MonitoringServiceListener` será notificada com a atual posição (padrão de projeto *Observer*). O monitoramento ocorrerá até que o método `removeMonitor(UUID clientID)` seja acionado.

A assistência durante a rota é provida pelo subcomponente `GuidanceService`. O início da assistência se dá com a execução do método `startGuidance`, que por sua vez: i) carrega objetos que representam a origem e o destino, podendo fazer uso, para tal fim, do `GeocodingService`; ii) busca a rota através do subcomponente `RoutingService`; iii) inicializa e registra objeto encarregado de armazenar as informações referentes à jornada do usuário (`Journey`); iv) inicia o monitoramento do usuário através do `MonitoringService`; v) registra a implementação do *listener* `GuidanceServiceListener`, associado-a ao identificador do usuário. O *listener* será notificado sempre que houver uma nova instrução disponível.

Para receber as notificações providas pelo `MonitoringService`, o `GuidanceService` implementa a interface `MonitoringServiceListener`. Dessa forma, ao ser notificado sobre a mudança de posição do usuário, o

[7] `https://jgrapht.org/`

`GuidanceService`: i) atualizará o objeto `Journey`, isso inclui descobrir os *beacons* vizinhos a posição atual (anterior e próximo) e calcular o azimute do trecho atualmente percorrido; ii) calculará a próxima instrução, levando em conta a diferença entre o azimute do próximo trecho e o azimute calculado no passo anterior; iii) notificará a implementação do *listener* `GuidanceServiceListener`, registrada anteriormente. As seguintes informações serão repassadas nesse último passo: instrução direcional, descrição da localização atual, o azimute do próximo trecho e um booleano que indica se o destino foi alcançado.

O cálculo do azimute é peça chave para o correto direcionamento do usuário. Este trabalho pressupõe que a orientação do usuário durante o trajeto é igual ao azimute do trecho atualmente percorrido. No início do trajeto, entretanto, considera-se que o usuário estará sempre voltado para o primeiro trecho. Para cálculo do azimute, utiliza-se a biblioteca GeographicLib[8] e as coordenadas geográficas dos pontos envolvidos. O algoritmo implementado na biblioteca foi descrito em [58]. A Figura 3.4 ilustra como o ângulo formado pela diferença entre os azimutes do trecho percorrido e do trecho posterior é convertido em uma instrução direcional.

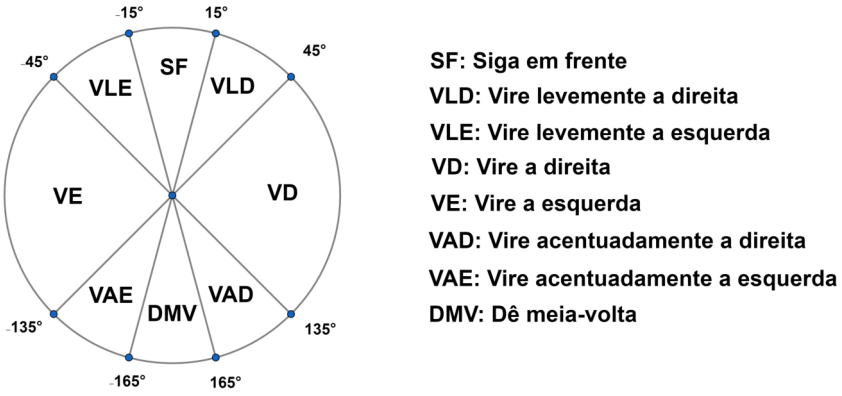

Figura 3.4: Instruções direcionais.

Por fim, o `ProtocolTranslatorService` resolve as requisições realizadas via *Websocket*, direcionando-a ao componente correto. Com o objetivo de manter a semelhança das requisições via *Websocket* de requisições a serviços REST, um protocolo baseado no HTTP (`NavigationServiceProtocol`) foi desenvolvido. O `NavigationServiceProtocol` consiste dos seguintes campos: URI (*Uniform Resource Identifier*), para identificar o recurso desejado (por exemplo, /instructions ou /buildings/id/places); *verb*, que indica a ação a ser executado no recurso (isto é,

[8] `https://geographiclib.sourceforge.io/`

GET, *POST*, *PUT* e *DELETE*); status, que indica a situação da requisição (por exemplo, *OK[200]*, *BAD REQUEST[400]*, *NOTFOUND[404]*, *INTERNAL SERVER ERROR[500]* e *SERVICE UNAVAILABLE[503]*); clientID, que identifica o cliente; e, *data*, um conjunto de pares chave-valor para troca de informações adicionais.

Para fazer uso dos serviços disponibilizados pelo `ProtocolTranslatorService`, a classe responsável por implementar a interface de comunicação do protocolo *Websocket* (`ServiceWebSocket`) implementa também a interface do *listener* `ProtocolTranslatorServiceListener`. A idéia por trás dessa abordagem é permitir que mensagens geradas como resposta às solicitações do usuário sejam enviadas. Ao receber requisições, através do método `callService(NavigationServiceProtocol nsProtocol)`, o `ProtocolTranslatorService` iniciará *thread* para processamento assíncrono da mesma. De fato, o processamento será executado pelo método `process(NavigationServiceProtocol request)`, que além de validar os dados da requisição, direciona-a ao serviço solicitado. Os campos URI e *verb* do protocolo `NavigationServiceProtocol` são utilizados para o direcionamento. Caso algum erro ocorra durante o processamento, será enviada mensagem de retorno com o status e a descrição correspondente. A descrição do erro estará no conjunto *data* sob a chave *message*. Em tese, todas as solicitações devem receber, no mínimo, uma mensagem de resposta.

A Figura A.3, presente no Apêndice A, exemplifica de forma sucinta a interação entres alguns do subcomponentes, durante a solicitação e a execução da funcionalidade de assistência em rota.

Inicialmente, a aplicação interage com o *Context consumer* repassando as informações necessárias (passo 1). O *Context consumer* monta a requisição no formato do protocolo `NavigationServiceProtocol` e a envia ao servidor (passo 1.1). A mensagem é recebida pela classe `ServiceWebSocket`, em seguida, é encaminhada para o `ProtocolTranslatorService` (passo 1.1.1). Este, por sua vez processa os campos URI e *verb* do `NavigationServiceProtocol` e os dados para solicitação e aciona o serviço correspondente (passos 1.1.1.1 e 1.1.1.1.1). O serviço de assistência é iniciado, conforme descrito em parágrafo anterior (passos 1.1.1.1.1.1, 1.1.1.1.1.2 e 1.1.1.1.1.3). Em caso de sucesso ou de erro, o `ProtocolTranslatorService` envia mensagem corresponte para a aplicação

cliente, refazendo o caminho inverso ao da solicitação (passos 1.1.1.2, 1.1.1.2.1 e 1.1.1.2.1.1). Uma vez iniciado o serviço, o `MonitoringService` consultará o *Context manager* e notificará o `GuidanceService` tão logo detecte a mudança de posição do usuário (passos 2 e 3). Então, o `GuidanceService` calculará a instrução direcional e enviará ao `ProtocolTranslatorService` (passos 3.1 e 3.2). A mensagem será repassada até chegar a aplicação cliente (passos 3.2.1, 3.2.1.1, 3.2.1.1.1 e 3.2.1.1.1.1).

A implementação dos subcomponentes seguem contratos estabelecidos por meio de interfaces. A separação entre a definição e a implementação propriamente dita permite a adoção do padrão de projeto Injeção de Dependência (*Dependency Injection* - DI). Injeção de dependência é um padrão que permite ao programador injetar objetos em uma classe usando um contêiner que é configurado externamente (geralmente um arquivo XML ou no próprio código da aplicação), em vez de permitir que a classe instancie diretamente os objetos [59]. O padrão mantém um baixo nível de acoplamento entre os diferentes subcomponentes do sistema, aumentando a reusabilidade e facilitando a manutenção e a execução de testes. A implementação do padrão deu-se com o uso do *framework* Guice[9].

3.2.4 *Context consumer*

O *Context consumer* provê meios para o uso de informações contextuais e serviços disponibilizados pela infraestrutura de *software*, apoiando a construção de novas aplicações. Localiza-se na camada de aplicação. A biblioteca foi desenvolvida em duas linguagens: em Javascript, voltada ao desenvolvimento de aplicações Web, e em Java, como um módulo Android.

Quanto ao protocolo utilizado para a comunicação entre aplicação cliente e servidor, em ambas as implementações adotou-se o padrão de projeto *Strategy*. A classe abstrata `ServiceConsumerStrategy` contém a assinatura dos métodos que representam as consultas disponibilizadas pela infraestrutura de *software*. Não existe classe correspondente a esta na implementação Javascript. Atualmente, existe uma classe concreta: a classe que implementa a comunicação via protocolo *Websocket* (`WebSocketConsumer`). Deu-se o nome de `Inavigs` (`inavigs`, em Javascript) para a classe exposta aos desenvolvedores.

[9] `https://github.com/google/guice`

Eventos e *Promises* são utilizados na implementação em Javascript. A união dessas duas abordagens possibilita a escrita de código assíncrono de forma mais legível. Um *Promise* é uma implementação de mônada e representa a eventual conclusão (ou falha) de um processamento assíncrono, bem como, seu valor resultante [60]. Mônada é um conceito oriundo da programação funcional e o seu uso permite o encadeamento de operações e o repasse do resultado calculado anteriormente para métodos subsequentes. Com base nesse conceito, códigos assíncronos com certa relação de dependência acabam residindo na mesma região, facilitando a leitura e a identificação dos mesmos. Eventos, por sua vez, denotam acontecimentos observáveis que ocorram no âmbito do *Document Object Model* (DOM). Eles podem surgir, por exemplo, da interação do usuário com a aplicação (`UIEvent`) ou programaticamente (`CustomEvent`). Na biblioteca, os eventos possuem o papel de notificar os métodos a serem executados após o término de computação assíncrona (*listeners*). São gerados programaticamente durante o recebimento de mensagens da infraestrutura e, cada evento é representado por um objeto baseado na interface `CustomEvent`.

Com o propósito de manter a compatibilidade com o M-Hub (Android API *level* 23), a implementação da biblioteca em Java, por outro lado, utilizou-se de *callbacks* para a execução de código após chamadas assíncronas. Mônadas foram adicionadas no Java 8 e são suportadas nativamente a partir da Android API *level* 24. Apesar disso, a estruturação do código mantém similaridade com a versão em Javascript.

Para o uso, o fluxo de trabalho a ser seguido, de forma geral, se resume a adição da biblioteca ao projeto, a configuração de parâmetros (por exemplo, o protocolo, as informações que identificam o servidor e a versão da API) e a inicialização da biblioteca chamando o método `start(UUID clientUUID)`. No Java, o método para configuração dos parâmetros é o `init(Properties properties)`, no Javascript, é o `setOptions(options)`. Ao ser invocado o método `start` armazena o identificador do cliente para uso posterior, inicializa objeto interno que implementa o protocolo de comunicação escolhido e, caso necessário, estabelece a conexão. Após a execução dessa função, qualquer serviço disponibilizado pela API poderá ser utilizado.

As listagens B.7 e B.8 do Apêndice B demonstram exemplos do código necessário para configuração, em Java e Javascript, repectivamente. É importante notar

que o código a ser executado após o término do método `start` (linhas de 10 a 20 na Listagem B.7 e 16 a 20 na Listagem B.8), em caso de sucesso ou erro, localiza-se em seguida ao mesmo, dando a idéia de causa e efeito.

Conforme mencionado anteriormente, desde que a API seja inicializada corretamente, qualquer serviço poderá ser solicitado. A Figura A.4 - Apêndice A demonstra a interação entre a aplicação, a classe `Inavigs` (representando o *Context consumer*) e o *Context reasoner*, ocorrida em resposta à solicitação do serviço de assistência durante a rota. Apesar de demonstrar a solicitação de um serviço específico, a sequência descrita a seguir pode ser estendida para qualquer serviço, em ambas as linguagens, respeitando as devidas proporções. Ao receber a solicitação, a classe `Inavigs` a repassará ao *Context reasoner* e, em seguida, registrará o *listener* responsável pelo tratamento da resposta (passos 1, 1.1 e 1.2). A requisição será codificada de acordo com o protocolo `NavigationServiceProtocol`. Ao receber a resposta, a classe `Inavigs` repassará os dados da mesma para o método: `onSuccess`, em caso de sucesso; e, `onFailure`, em caso de erro (passos 2, 2.1 e 2.3). Ao final, caso não existam mais mensagens a serem recebidas, a classe `Inavigs` removerá o *listener* (passo 2.3).

A Listagem B.9 - Apêndice B exibe um exemplo de código em Java para inicialização do serviço de assistência em rota. Nesse exemplo, o *listener* foi implementado como uma váriavel (linhas de 2 a 20) ao invés da abordagem utilizada na Listagem B.7, cuja implementação ocorreu na própria chamada do método. Um mapa (linha 7) será utilizado para repassar as informações recebidas da infraestrutura para o método que será executado a cada instrução recebida (linhas de 7 a 15). As linhas 12 e 13, em especial, demonstram como a instrução direcional pode ser extraída do mapa. A linha 19 contém a assinatura do método que será executado em caso de erro. Este, recebe o código do status e a mensagem correspondente ao erro reportado pela infraestrutura. Caso o erro ocorra localmente, o código do status será nulo. As linhas 25 e 26 exibem a chamada ao método de assistência em rota. Para esse método, são repassados como parâmetros de entrada: a origem, o destino, um booleano que indica se o caminho deve ser acessível e o *listener*. O exemplo pode ser estendido aos outros serviços disponibilizados pela API.

A Listagem B.10 - Apêndice B exibe código em Javascript com o mesmo objetivo da listagem anterior (B.9). As linhas de 2 a 10 descrevem a função a ser executada no caso de recebimento de mensagem. Basicamente, ela recebe como

parâmetro um objeto `event` baseado na interface `CustomEvent`. A linha 8 demonstra como a instrução direcional pode ser obtida. É importante notar o caminho para acesso à referida informação (`event.detail.data`). O atributo `detail` é oriundo da própria interface e serve para armazenar qualquer informação passada durante a inicialização do evento. O objeto `data`, por sua vez, representa o mesmo mapa (`map`) presente na linha 7 da Listagem B.9. Não é coincidência, portanto, o fato do nome do atributo ser igual em ambos os códigos (`instruction`). As linhas de 13 a 15 exibem a função a ser executada em caso de erro. O parâmetro também é um *event* e o acesso a mensagem de erro se dá de forma semelhante ao relatado anteriormente. As linhas de 19 a 24 demonstram a invocação da função de assistência em rota. Nesse caso, os parâmetros são: a origem, o destino, um booleano que indica se o caminho deve ser acessível, a função a ser executada no caso de recebimento de mensagem e a função a ser executada em caso de erro. Este é o único caso onde uma função da API Javascript precisa receber como parâmetro, obrigatoriamente, outra função para tratar do recebimento de mensagens. Essa singularidade é derivada do próprio serviço, visto que o mesmo é contínuo. Em outras palavras, uma vez requisitado, o serviço enviará mensagens até que o cancelamento seja explicitamente solicitado. Os outros serviços, em comparação, são do tipo requisição-resposta e o modelo adotado com *Promise* é ideal nesse sentido.

3.3 Limitações do INavigS

Nesta seção, serão apresentadas resumidamente algumas limitações do INavigS, bem como, as medidas que devem ser executadas para superar as mesmas.

O fato do *Context manager* ser capaz de lidar apenas com a posição do usuário representa uma limitação desta infraestrutura. Além dos dados de contexto que podem ser capturados pelo *Context provider*, informações relacionadas com o uso da infraestrutura também poderiam ser armazenadas (por exemplo, destino). Uma vez disponíveis, essas informações poderiam apoiar a construção de um componente de recomendação, bem como, a construção de aplicações mais robustas.

Outra limitação, em parte relacionada à anterior, é quanto a orientação do usuário. Na implementação do serviço responsável pela assistência em rota

pressupõe-se que a orientação do usuário é: a) igual a do primeiro trecho a ser percorrido, no início do deslocamento; b) a mesma do trecho que ele atualmente percorre, durante o deslocamento. Essas suposições podem, em muitos casos, não refletir a realidade. No pior cenário, isso levará a desvio desnecessários, aumentando o tempo e a distância total percorrida pelo usuário. Ademais, sem considerar as devidas adequações realizadas pelo desenvolvedor, a infraestrutura será inadequada para deficientes visuais, uma vez que estes são muito afetados por questões relacionadas à orientação.

A inexistência de consulta que retorne as últimas localizações com a data em que foram registradas ou a última localização conhecida, independentemente do tempo em que ela foi registrada, ou ainda de mecanismo de subscrição para notificar a alteração da localização de um usuário, no *Context manager*, gera a necessidade do *Context provider* enviá-la periodicamente. Relembrando que, no que concerne ao *Context provider* e ao *Context manager*, a localização é representada como o encontro entre um dispositivo móvel e um *beacon* BLE. Caso o envio não fosse necessário, o ganho energético no dispositivo móvel seria ainda maior.

Para sobrepujar as três limitações, faz-se necessário estender o *Context manager*, generalizando o tratamento, o armazenamento e a disponibilização das informações contextuais. Quanto à obtenção da orientação do usuário, um passo adicional deve ser realizado: a adição da implementação do protocolo S2PA que trata de informações provenientes dos sensores internos dos dispositivos móveis à versão do *middleware* M-Hub utilizada por este trabalho. Dessa forma, a orientação do usuário pode ser inferida a partir de dados do magnetômetro (isto é, a bússola), do giroscópio ou de ambos, por exemplo. É importante ressaltar que a referida implementação do protocolo S2PA já existe em outras versões do M-Hub.

3.4 Considerações Finais

Neste capítulo foi apresentada a infraestrutura de *software* INavigS, proposta por este trabalho. Foram descritos cenários de aplicação, a arquitetura e suas camadas, os elementos que a compõem e algumas de suas limitações. Além disso,

descreveu-se alguns aspectos da implementação, bem como, de que forma alguns dos problemas provenientes da navegação *indoor* foram abordados.

O INavigS visa fornecer suporte ao desenvolvimento de aplicações que façam uso de serviços de navegação *indoor*, tais como: geocodificação, cálculo de rota, informações acerca do espaço interno e assistência durante a rota. Espera-se que a partir desses serviços, aplicações com maior valor agregado possam ser construídas, rapidamente, nos mais diversos domínios.

4 Trabalhos Relacionados

Ao longo do últimos anos, diversos pesquisadores voltaram seus esforços para a proposição de Sistemas de Navegação *Indoor* [61] [62] [63] [64]. Entretanto, poucos trabalhos fizeram uso de *beacons* BLE como fonte principal ou auxiliar na obtenção do posicionamento *indoor*. Como mencionado anteriormente, o BLE surge como um alternativa viável frente as demais, devido ao baixo custo, facilidade de implantação e o amplo suporte dado à tecnologia por dispositivos móveis.

Este capítulo apresenta de forma sucinta alguns desses trabalhos, ressaltando aspectos da arquitetura, disposição dos *beacons*, representação do espaço e métodos utilizados. No fim, apresentará uma comparação entre as diferentes abordagens e a proposta por este trabalho.

4.1 GuideBeacon

GuideBeacon [65] é um sistema de navegação *indoor* para deficientes visuais e desorientados que faz uso de *beacons* BLE implantados no ambiente e pode ser operado em qualquer *smartphone* convencional. Possui a funcionalidade de pré-visualização da rota, em adição à funcionalidade de navegação.

Além dos *beacons*, o sistema possui outros três componentes: *Beacon Manager*, *Map Database* e GuideBeacon app. O *Beacon Manager* é o encarregado de responder consultas relacionadas a pontos de interesse no espaço *indoor*, serviço tipicamente fornecido pela plataforma do fabricante de *beacons*. A consulta ao *Beacon Manager* ocorre durante a seleção do destino. Mapas são fornecidos pelo componente *Map Database* e, após o *download*, todo o processamento ocorre no dispositivo móvel, mais precisamente, no GuideBeacon app. Logo, o GuideBeacon app é uma aplicação Android que engloba funções de posicionamento, navegação e orientação, tornando-se o principal componente desta arquitetura.

Beacons são posicionados, inicialmente, em cada ponto de interesse e em cada interseção. Em seguida, cada trecho em desacordo com o requisito da distância

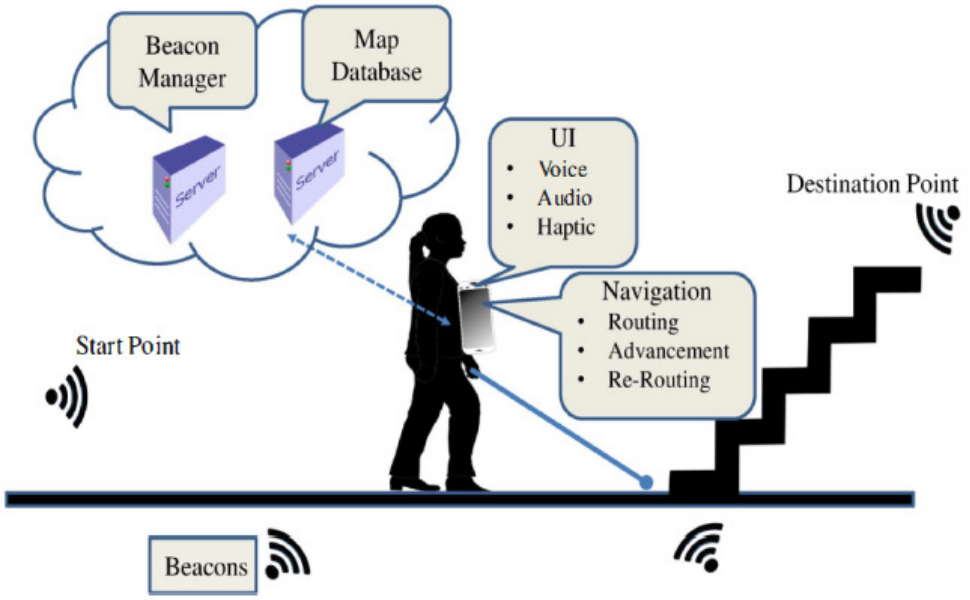

Figura 4.1: Arquitetura do GuideBeacon [65].

mínima entre *beacons* receberá dispositivos adicionais. Para cálculo da posição foi utilizado um algoritmo de detecção de proximidade. A ideia é calcular a média móvel ponderada sobre o conjunto de medições RSSI associadas a cada *beacon* e avaliar se esse valor é superior a um certo limiar. Realiza-se a operação supracitada apenas para os *beacons* que atingiram um número mínimo de observações dentro do dobro de tempo necessário para que essas obsevações aconteçam. Para ser considerado o *beacon* mais próximo, além de satisfazer as duas condições anteriores, deve possuir dentro do seu conjunto de observações, um número mínimo de registros cujo RSSI seja superior a um segundo limite.

O ambiente interno é modelado como um mapa 3D, confeccionado manualmente e representado como uma *string*. Após o *download*, o grafo sobre o qual o Dijkstra será executado é extraído do mapa. O peso atribuído ao grafo é a distância, portanto, o resultado do Dijkstra será a rota mais curta. A bússola é utilizada para auxiliar no direcionamento do usuário durante o percurso. E, caso o usuário alcance um *beacon* que não esteja na rota proposta, o re-roteamento será executado.

A entrada de dados pode ocorrer por comandos de voz ou por toque. No primeiro caso, a voz do usuário é capturada e, posteriormente, convertida em texto através da classe `SpeechRecognizer` do Android. Essa função será utilizada principalmente ao consultar destinos. Objetivando manter apenas uma das mãos ocupadas, toda a interface foi projetada para permitir que o usuário dê comandos

de voz após dois toques na tela. Por padrão, as respostas às consultas efetuadas e as instruções de navegação são também apresentadas por mensagens de voz. Para geração dessas mensagens, faz-se uso da API *Text-To-Speech* do Google. Ao fornecer instruções de navegação, a confirmação da orientação a ser seguida é realizada por meio de vibração e bipes. O usuário pode desligar a entrada e a saída por voz, caso necessite. Nesse caso, a entrada será por toque e a saída por texto.

4.2 InLoc

InLoc [66] é um sistema de posicionamento e rastreamento robusto, voltado à ambientes *indoor*, que faz uso de dipositivos móveis e inclui funcionalidade para encontrar rotas.

O sistema se divide em dois componentes básicos, como exposto na Figura 4.2: um servidor de localização e uma aplicação executada em dispositivo móvel. A aplicação móvel possui, dentre as suas funções, a de escanear *beacons* e a de estimar a orientação e o tamanho do passo. O servidor de localização, por sua vez, realiza a maior parte das tarefas relacionadas com o posicionamento, roteamento e provimento dos mapas.

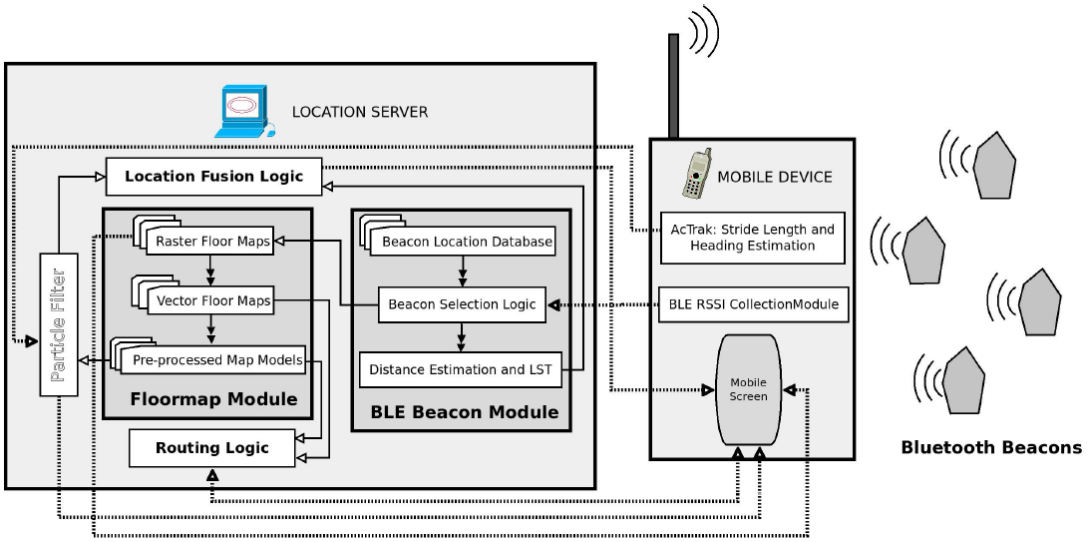

Figura 4.2: Arquitetura do InLoc [66].

O InLoc utiliza DR como fonte primária de posicionamento, mediante a aplicação do algoritmo filtro de partículas. O algoritmo de filtro de partículas é uma forma não-paramétrica de estimação Bayesiana, comumente usada em visão

computacional e rastreamento, para cálculo da localização [67]. *Beacons* são utilizados para derivação da posição inicial e para auxiliar no processo de obtenção de nova posição, quando o filtro de partículas falha em convergir de forma rápida. A orientação é calculada a partir da fusão entre os dados da bússola e os dados do giroscópio.

Quanto à distribuição dos *beacons*, o espaço é dividido em quadriláteros de diferentes tamanhos e em cada vértice um *beacon* é posicionado. A distância máxima entre *beacons* é de 8 metros. Para inferência da localização utilizando os *beacons*, primeiro seleciona-se os quatro com maior intensidade de sinal recebido e, caso eles não pertençam ao mesmo quadrilátero, o processo é repetido. Em seguida, utiliza-se o *K-Nearest Neighborg* (KNN) para estimar a distância entre o dispositivo e cada *beacon* do quadrilátero. De posse da distância e do posicionamento de cada *beacon*, aplica-se o método *Least Squares Trilateration* (LST) [68]. O LST é um algoritmo que estima a posição de uma objeto alvo com base nas medidas simultâneas de distância de vários pontos de referência, resolvendo uma formulação de trilateração não-linear de mínimos quadrados usando técnicas de álgebra linear padrão.

Em relação ao mapeamento do espaço, foi proposto um método para derivação de modelo que permita tanto a execução do filtro de partículas como a obtenção de rotas. Inicialmente, converte-se uma imagem *raster*, referente ao andar, para um conjunto de polígonos fechados usando técnicas de detecção de bordas. Após essa etapa, o espaço é dividido em um grid de células e, para cada uma delas, a validade é verificada. Células inválidas são aquelas cuja área coincide com a área de um polígono em mais de 50%. O modelo gerado é, enfim, adequado para roteamento. Para encontrar o menor caminho, foi utilizado o algoritmo Dijkstra. Nesse trabalho, não há acompanhamento durante a execução da rota.

4.3 NavCog

NavCog [69] é um sistema baseado em *smartphone* que fornece um assistente de navegação passo a passo para deficientes visuais. Além da navegação, o sistema pode fornecer ao usuário informações sobre pontos de interesse nas proximidades ou instruções de acessibilidade.

O NavCog é dividido em três componentes, conforme mostra a Figura 4.3: *Map Server*, que armazena as informações descrevendo cada ambiente e o modelo espacial requerido para localizar corretamente o usuário dentro desses ambientes; *beacons* BLE, instalados previamente no espaço e que, periodicamente, emitem o sinal utilizado para estimação da localização do usuário; e, o NavCog App, uma aplicação iOS que guia o usuário até o destino utilizando mapas obtidos a partir do *Map Server* e da posição calculada a partir dos *beacons*. Feito o *download* do mapa, todo o processamento ocorre no dispositivo móvel. Com o objetivo de encurtar o tempo e o custo para implantação do sistema em outros ambientes, foi desenvolvida ferramenta Web para auxiliar na construção e edição de mapas.

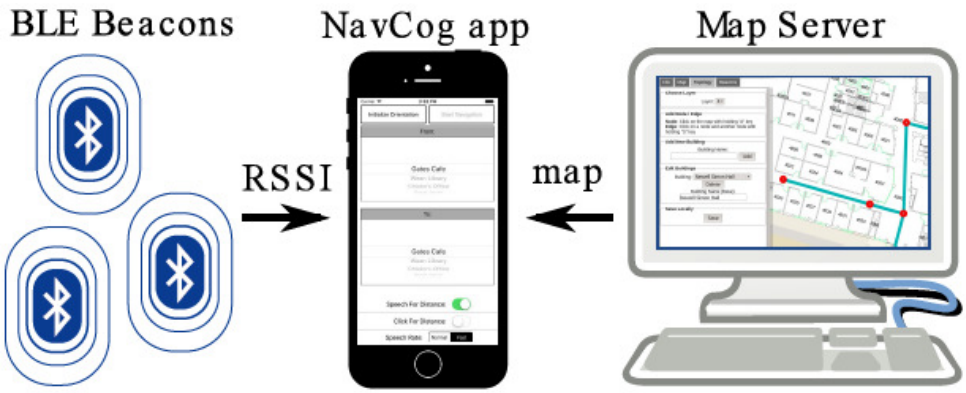

Figura 4.3: Arquitetura do NavCog [69].

Os *beacons* são distribuídos ao longo do ambiente em um intervalo pré-definido e, preferencialmente, são posicionados em paredes opostas de forma alternada. É utilizado o algoritmo KNN para implementar a técnica *Fingerprint*, alcançando a acurácia de 1.5 m. A orientação é obtida a partir dos dados do giroscópio. Em trabalho posterior [70], os autores utilizaram filtro de partículas para integrar DR e BLE, melhorando a acurácia do sistema para 0.68 m. Além disso, avaliaram como a quantidade e a densidade dos *beacons* e dos exemplos de treinamento impactam no balanço entre a acurácia da localização e o custo de configuração do ambiente.

Em relação a representação do espaço, o NavCog utiliza um grafo de pontos de referência (isto é, pontos de decisão e de interesse). O ângulo entre as arestas são armazenados e servem para fornecer direções precisas durante a navegação. O algoritmo Dijkstra foi utilizado para encontrar o menor caminho.

Três tipos de mensagens podem ser dadas durante a navegação: anúncios de distância, na forma de mensagens de voz sintetizadas ou alertas sonoros; instruções

de ação, que se dividem em instruções de direção e instruções de trânsito (por exemplo, durante mudanças de nível ou entre espaços *indoor* e *outdoor*); e descrições de pontos de interesse. As mensagens do último tipo são opcionais, exceto a que é disparada quando o usuário chega ao nó de destino. Uma limitação do NavCog é o fato deste não informar ao usuário no caso de desvio em relação à rota proposta.

4.4 StaNavi

O StaNavi [71] é um sistema de navegação *indoor*, voltado para deficientes visuais, que fornece instruções de voz passo a passo em grandes e complexas instalações. Foi implementado na Estação de Tóquio, uma das mais movimentadas estações de trens do mundo. As três principais funcionalidades são: descobrir a atual localização; navegar livremente, obtendo informações acerca de pontos de referência nas proximidades da localização atual; e navegação passo a passo. Além disso, outras características do sistema podem ser destacadas: interface que permite a interação com apenas uma das mãos, pré-visualização da rota e dicas de navegação que descrevem caminhos complexos ou espaços abertos.

A arquitetura do StaNavi, conforme ilustrada na Figura 4.4, é composta por três elementos: aplicação que roda no dispositivo móvel do usuário, infraestrutura de *beacons* e servidor baseado em nuvem, responsável por computar as rotas entre origem e destino.

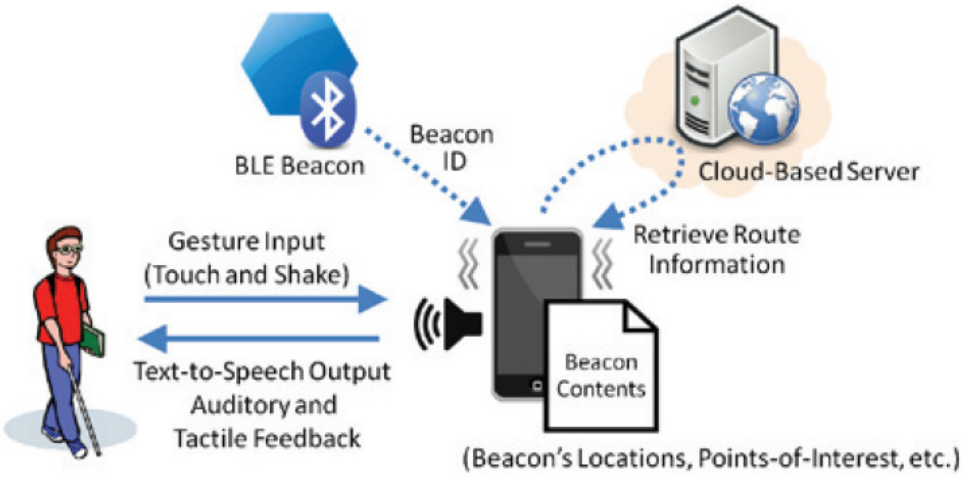

Figura 4.4: Arquitetura do StaNavi [71].

Os *beacons* foram distribuídos em locais próximos a pontos de referência (por exemplo, plataformas, saídas, banheiros e elevadores). A localização do usuário é obtida a partir da técnica de localização por proximidade e a orientação por meio da bússola presente no dispositivo móvel. A busca por *beacons* ocorre a cada dois segundos. Como mencionado anteriormente, rotas são calculadas no servidor. Ao receber a solicitação via HTTP, o servidor calcula a menor rota usando o algoritmo de roteamento Dijkstra e, posteriormente, retorna as informações em um arquivo XML. Com exceção do cálculo de rotas, todo o processamento ocorre no dispositivo móvel.

Por meio de mensagens de voz, o sistema fornece instruções com alto grau de detalhe. A localização é apresentada em dois níveis de hierarquia: área e ponto de interesse. A área é uma parte significativa do espaço (por exemplo, salões de espera e corredores). Pontos de interesse representam localizações importantes que podem ser identificadas através de *beacons*. Em relação às mudanças de direção, foram utilizadas quatro direções absolutas e, a partir delas, foi utilizado um sistema baseado em horas para fornecer direções diagonais. Caso, dentro de 30 segundos, *beacons* que não estejam presentes na rota sejam encontrados, o usuário é notificado sobre o desvio em relação ao caminho sugerido.

4.5 WheelScout

WheelScout [72] [73] é uma aplicação para cálculo de rotas *outdoor*/*indoor* acessíveis para cadeirantes, customizadas de acordo com o perfil do usuário. A aplicação permite a adição de barreiras permanentes ou temporárias no mapa, salvar as localizações favoritas e a interação multimodal (por exemplo, teclado, toque ou reconhecimento de comandos de voz).

Como a aplicação foi desenvolvida em HTML5, as funções da API de geolocalização foram utilizadas para obtenção da posição em ambientes *outdoor*. Então, caso acessada por dispositivo móvel, a posição GPS atual será enviada. Quando acessada em dispositivos sem GPS, a posição é estimada via endereço IP e sinais Wi-Fi disponíveis. Utiliza-se a tecnologia BLE, somado ao método de trilateração, em ambientes *indoor*. De forma oportuna, o UUID de cada *beacon* é gerado a partir das coordenadas geográficas e do andar em que se localizam. Essa abordagem permite

que a trilateração seja executada apenas com a decodificação dos UUIDs, ou seja, sem a necessidade de realização de consulta em base de dados. Para aumentar a precisão, o cálculo da trilateração é executado múltiplas vezes a cada segundo e, então, a média é considerada como a posição do usuário.

Os mapas *indoor* foram desenhados na ferramenta Java OpenStreetMap Editor e, após modelagem da estrutura interna, áreas transitáveis são definidas por meio de pontos de referência. Tais pontos são utilizados para geração da rotas, em momento posterior. Utiliza-se dados geoespaciais do projeto de mapeamento colaborativo OpenStreetMap para obtenção de informações do espaço externo. Em ambos os casos, o algoritmo Dijkstra é utilizado para encontrar rotas sobre um grafo direcionado, cujo peso é a distância em quilômetros. As rotas são exibidas utilizando a metáfora do semáforo. Rotas livre de barreiras são criadas na cor verde, rotas marcadas em amarelo possuem barreiras que podem causar dificuldades para certos usuários e rotas em vermelho são intransponíveis.

Como a rota e a posição do usuário é exibida no mapa, espera-se que o usuário seja capaz de alcançar o destino através de comparação visual, portanto, não faz parte do escopo deste trabalho o fornecimento de instruções sob medida durante a execução da rota e nem a notificação caso o mesmo se afaste do caminho sugerido.

4.6 Análise Comparativa

Características das diferentes abordagens apresentadas neste capítulo são comparadas na Tabela 4.1. Os critérios adotados para avaliação derivam dos desafios inerentes ao desenvolvimento de aplicações que possuem serviços de navegação dentre as suas funcionalidades. São eles:

- **Obtenção da localização e orientação do usuário**: analisa a tecnologia e o método empregado, incluindo aspectos relacionados com a acurácia, a facilidade de implantação em outros espaços, a complexidade do processamento e o tempo necessário para a disponibilização da informação (isto é, compreende o tempo de processamento e o tempo de tráfego na rede, caso necessário).

- **Representação do espaço *indoor***: avalia se o modelo espacial é capaz de fornecer meios para extração de rotas, informações adicionais sobre os espaços e a facilidade de extensão para outros locais.

- **Geração de rotas customizadas**: analisa a capacidade do sistema de gerar rotas ajustadas às necessidades do usuário, bem como, o nível de customização permitido.

- **Assistência ao longo da rota**: verifica se o sistema é capaz de fornecer instruções ao longo da rota, incluindo a capacidade de informar e adaptar-se aos desvios indesejados.

- **Gerenciamento de energia**: considera a existência de mecanismos com o propósito de salvaguardar a energia dos dispositivos móveis, durante a execução dos serviços.

- **Diponibilização de API**: visa analisar os mecanismos oferecidos a fim de apoiar o desenvolvimento de novas aplicações.

.

4.6.1 Obtenção da localização e orientação do usuário

Em relação às tecnologias utilizadas para a obtenção da localização e da orientação, a maioria dos trabalhos fez uso de ambas as tecnologias BLE e IMU. O WheelScout e o INavigS são a exceção, visto que o primeiro fez uso do GPS, além do BLE, e o segundo utiliza apenas o BLE. Dentre aqueles que utilizam BLE e IMU, apenas o InLoc utiliza o IMU como tecnologia principal.

Quanto ao método e acurácia, o InLoc adotou DR. Nesse caso, os *beacons* assumiram papel auxiliar e são utilizados para a obtenção da posição inicial e durante a recuperação de erros e, em ambos os casos, dois métodos são empregados: KNN (*Fingerprint*) e LST. Neste trabalho, o erro médio na localização foi de 0.9 metros. Na implementação do *Fingerprint*, o NavCog também utilizou o KNN, alcançando a acurácia de 1.5 metros. O WheelScout implementou um método de trilateração que, de forma oportuna, utiliza as coordenadas a partir da decodificação do UUID. O GuideBeacon, StaNavi e INavigS implementam métodos

Tabela 4.1: Análise comparativa dos trabalhos relacionados

		GuideBeacon [65]	**InLoc [66]**	**NavCog [69]**	**StaNavi [71]**	**WheelScout [72] [73]**	**INavigS**
Público-alvo		Deficientes visuais e desorientados	Público em geral	Deficientes visuais	Deficientes visuais	Cadeirantes	Público em geral e cadeirantes
POSICIONAMENTO	Tecnologia	BLE; IMU	IMU; BLE	BLE; IMU	BLE; IMU	GPS; BLE	BLE
	Método	Proximidade	DR, Fingerprint e Trilateração	Fingerprint	Proximidade	Trilateração	Proximidade
	Processamento	Cliente	Cliente/Servidor	Cliente	Cliente	Cliente	Cliente/Servidor
	Intervalo entre *scans*	–	–	–	Fixo (2 segundos)	–	Dinâmico
	Disposição dos *beacons*	pontos de interesse (POIs), interseções e espaçados	Quadriláteros com lados de até 8 metros	A cada metro, alternando os lados	POIs	–	Um a cada 3 metros
ROTAS	Algoritmo	Dijkstra	Dijkstra	Dijkstra	Dijkstra	Dijkstra	Dijkstra
	Customização	Não	Não	Não	Não	Sim	Sim
	Critério	Distância	Distância	Distância	Distância	Acessibilidade e distância	Acessibilidade e distância
Modelo espacial		Mapa 3D e grafo de sensores	Grid	Grafo de pontos de referência	Hierárquico e grafo de sensores	Geométrico e grafo de pontos de referência	Simbólico e grafo de sensores
Assistência em rota		Sim	Não	Sim	Sim	Não	Sim
Feedback		Voz, texto, tátil e aviso sonoro	Mapa	Voz e mapa	Voz, texto, tátil e aviso sonoro	Mapa	Texto
Forma de comunicação		*Pull*	*Pull*	*Pull*	*Pull*	*Pull*	*Pull* e *Push*
API		Não	Não	Não	Não	Não	Sim

de localização por proximidade. Em particular, o GuideBeacon criou uma versão objetivando mitigar o impacto da variabilidade nas medições de RSSI, visto que realiza escaneamento contínuo. StaNavi e INavigS, em contrapartida, possuem intervalos entre escaneamentos e, dessa forma, processam a informação levando em conta o intervalo em que a fase de escaneamento encontra-se ativa. WheelScout, GuideBeacon e StaNavi não informam sobre a acurácia do sistema.

No que se refere à orientação, GuideBeacon e StaNavi utilizam a bússola presente no dispositivo móvel. NavCog utiza a informação oriunda do giroscópio. O InLoc realiza a fusão entre os dados da bússola e os dados do giroscópio. A fusão se justifica em razão dos distúrbios magnéticos que a bússola pode vir a sofrer no ambiente *indoor*, bem como, as distorções sofridas pelo giroscópio. A ideia chave por trás da fusão é que a bússola e o giroscópio demonstram características de variação semelhantes apenas quando não há distúrbios magnéticos externos. O WheelScout não faz menção ao cálculo da orientação. O INavigS pressupõe que a orientação do usuário é a mesma do trecho que percorre ou, no caso do usuário estar no início do trajeto, a mesma do trecho inicial.

Com relação à complexidade do processamento e o tempo necessário para a disponibilização da informação, não existem dados que permitam uma avaliação quantitativa. O InLoc e NavCog empregam métodos cujo custo computacional pode ser médio ou alto [48]. Uma diferença entre os dois é quanto ao local do processamento, já que o InLoc realiza o processamento no servidor, enquanto o NavCog realiza o processamento no dispositivo móvel. Ambos não fazem qualquer menção sobre o tempo necessário para o processamento e diponibilização da informação. O método de trilateração empregado no WheelScout não é computacionalmente custoso, entretanto, é executado múltiplas vezes para redução do erro. Isso pode elevar o tempo necessário para disponibilização da informação. A complexidade inserida no método de localização por proximidade criado pelo GuideBeacon conjuntamente com o intervalo de transmissão de anúncios dos dispositivos BLE podem influenciar negativamente no tempo de processamento. O StaNavi e o INavigS adotaram um método relativamente simples, mas, em ambos, deve-se considerar o intervalo entre escaneamentos no cálculo do tempo total de processamento. Em relação ao INavigS, deve-se considerar, adicionalmente, o tempo necessário para o tráfego das informações na rede, em razão de parte do processamento ocorrer na nuvem.

No tocante à expansibilidade do método para novos espaços, o InLoc e o NavCog, por implementarem o *Fingerprint*, necessitam de uma fase de preparação *offline* e, esta, pode ser bastante trabalhosa. Essa obrigatoriedade dificulta a implantação dos mesmos em novos espaços. WheelScout, GuideBeacon, StaNavi e INavigS, devido a simplicidade do métodos adotados, podem ser introduzido em novos ambientes com certa facilidade. Essa á a principal razão do INavigS ter adotado um método de localização por proximidade.

4.6.2 Representação do espaço *indoor*

Com relação à representação do espaço *indoor*, a estratégia adotada pelo GuideBeacon foi a criação de um mapa 3D. O mapa foi criado manualmente e foi representado como texto. Para a geração de rotas, faz-se necessário a extração de um grafo de sensores a partir desse modelo. A utilização de um mapa 3D permite a represenção do espaço *indoor* com um maior nível de detalhe, entretanto, o esforço necessário para a contrução do mesmo pode ser um impeditivo para a sua adoção em novos espaços.

O InLoc propôs um modelo em grid, útil para a extração de rotas e para a aplicação do algoritmo filtro de partículas. Objetivando permitir a escolha do destino, o mapa é pré-anotado com localizações chaves. Apesar disso, carece de informações que descrevam melhor o espaço. Nos testes executados pelo autor, o modelo apresentou um bom desempenho durante a execução do algoritmo Dijkstra. Adicionalmente, foram criadas ferramentas e um método com o objetivo de facilitar a adaptação do sistema a novos espaços.

O StaNavi criou um modelo hierárquico, dividido a nível de área, uma parte significativa do prédio, e a nível de pontos de referência que, por sua vez, representam localizações importantes que podem ser identificadas através de *beacons*. Para obtenção de rotas, utiliza-se um grafo de sensores. Devido a sua simplicidade é de fácil adoção em novos espaços.

O WheelScout utilizou um modelo geométrico enriquecido (por exemplo, tipo e condições do piso), baseado no projeto OpenStreetMap. As áreas transitáveis são definidas como um grafo de pontos de referência, possibilitando a extração de rotas com eficiência. Essa abordagem possibilita uma suave transição entre espaços

indoor e *outdoor* e a extração de rotas customizadas em tempo de execução. Além disso, o uso da ferramenta OpenStreetMap Editor facilita, em parte, a adoção do modelo em novos espaços.

O NavCog fez uso de um grafo de pontos de referência, construído através de ferramenta Web. O modelo é simples e fácil de ser adotado em novos espaços. A ferramenta Web permite ainda adicionar informações sobre POIs e visualizar a localização dos *beacons*.

O INavigS, enfim, fez uso de um modelo hierárquico em três níveis: prédio, elementos do prédio e sensores. Os níveis prédio e elementos do prédio são a base para informações semânticas. O nível de sensores é a base para a construção do grafo que possibilita a extração de rotas. A abordagem é simples, facilitando a adoção em novos espaços. Este, inclusive, é o princípio que regeu a construção desse modelo.

4.6.3 Geração de rotas customizadas

Apesar da enorme demanda por rotas customizadas, poucos sistemas possuem a capacidade de fornecê-las. É importante ressaltar a relação existente entre o nível de customização e o nível de detalhe do modelo espacial. Em geral, um maior nível de customização exigirá um modelo espacial mais rico, aumentando o esforço necessário para contruí-lo. Dentre os trabalhos apresentados, apenas o WheelScout, sistema cujo público-alvo são cadeirantes, permite a personalização das rotas levando em conta o tipo e a condição do piso, o tipo e as dimensões da cadeira de rodas e limitações individuais (por exemplo, número de degraus e inclinação máxima). O INavigS disponibiliza um grau de customização menor, permitindo apenas a geração de rotas para o público em geral e para cadeirantes. Portanto, optou-se por uma abordagem que facilite a construção do modelo espacial e, consequentemente, a adoção da infraestrutura em novos espaços.

4.6.4 Assistência ao longo da rota

Assistência durante a rota é uma importante funcionalidade que visa apoiar usuários durante o seu deslocamento até o destino desejado. O GuideBeacon, além de fornecer instruções, possui mecanismos para re-roteamento e a confirmação da direção

a ser seguida por meio de vibrações e bipes. O NavCog também guarnece usuários com intruções (isto é, anúncios de distância, instruções de direção e as que indicam mudança de nível ou entre ambientes *indoor* e *outdoor*), porém possui a limitação de não informar sobre desvios. Um aviso sonoro confirma a direção a ser seguida. StaNavi fornece instruções com alto grau de detalhe, aproveitando-se do seu modelo espacial hierárquico dividido em dois níveis. As direções são dadas a partir de quatro direções absolutas e, quando necessário, um sistema baseado em horas é utilizado para prover direções diagonais. Caso sejam detectados *beacons* que não fazem parte da rota durante 30 segundos, o usuário é notificado sobre o desvio.

Objetivando possibilitar a construção desses mecanismos, o INavigS fornece o azimute, ângulo formado entre o segmento de reta do próximo trecho e o norte geométrico, a descrição do local atual, um booleano que indica se a rota chegou ao fim e instruções direcionais em texto. Além disso, notifica o usuário sobre desvios, tão logo detecte um *beacon* que não pertença ao percurso sugerido. Apenas para exemplificar, o mecanismo para confirmação da direção a ser seguida pode ser construído com base no azimute e nos dados da bússola do dispositivo móvel.

4.6.5 Gerenciamento de energia

Nenhum dos trabalhos apresentados neste capítulo demonstrou preocupação em adicionar mecanismos para salvaguardar energia dos dispositivos móveis, durante a execução de seus serviços. Apesar disso, o StaNavi implementou intervalo de dois segundos entre *scans*, o que, de certa forma, é uma contribuição nesse sentido. O INavigS faz uso de intervalo dinâmico entre *scans*, regulado de acordo com o nível de bateria do dispositivo móvel. O pré-processamento dos dados através do serviço MEPA é outro mecanismo utilizado para salvaguardar energia. Ao invés de enviar todos os eventos de descoberta para processamento na nuvem, utiliza-se o motor CEP para a detecção do *beacon* com maior RSSI, em cada *scan*. Ambos, o serviço de pré-processamento e a adaptação do intervalo entre *scans*, são mecanismos oriundos do *middleware* M-Hub. Além disso, como a maior parte do processamento ocorre na nuvem, implementou-se a comunicação baseada no protocolo *Websocket* que, em comparação ao protocolo HTTP, consome uma quantidade menor de energia [74].

4.6.6 Diponibilização de API

Outro aspecto que vem sendo negligenciado é a disponibilização de APIs. Embora todos os trabalhos possuam serviços hospedado remotamente, não há informação alguma sobre a disponibilização destes para terceiros.

O INavigS, em contramão aos trabalhos descritos neste capítulo, foca-se na disponibilização dos serviços através de API. A ideia é viabilizar o desenvolvimento de novas aplicações, popularizando serviços de navegação *indoor*, visto que estes ainda não são utilizados em larga escala. Para isso, oferece biblioteca que expõe os serviços oferecidos aos desenvolvedores.

4.7 Considerações Finais

Este capítulo apresentou os principais trabalhos relacionados à navegação *indoor* que empregaram *beacons* BLE para auxiliar na obtenção do posicionamento. Os trabalhos GuideBeacon [65], InLoc [66], NavCog [69], StaNavi [71] e WheelScout [72] [73] foram dissecados e sua principais características expostas em detalhes. Após isso, os trabalhos foram comparados, sob critérios derivados dos desafios enfrentados durante o desenvolvimento de aplicações que fazem uso de serviços de navegação *indoor*, com a abordagem proposta neste trabalho.

Quanto à obtenção da localização e orientação do usuário, nenhuma abordagem destacou-se em todos os quesitos, reforçando a ideia de que é preciso avaliar as restrições e necessidades de cada sistema para projetar a solução ideal. Nesse quesito, o INavigS, sempre primando pela facilidade de implantação em novos espaços, propôs uma abordagem baseada em proximidade.

Acerca da representação do espaço *indoor*, todos os trabalhos são capazes de fornecer rotas e informações adicionais sobre os espaços. Apesar disso, é importante ressaltar que existe uma relação conflitante (*trade-off*) entre a expressividade do modelo e a complexidade exigida para a construção do mesmo. Dessa forma, a escolha do modelo ideal deve levar em conta os requisitos da aplicação. No INavigS, optou-se, mais uma vez, por uma abordagem de fácil implantação.

Em relação à geração de rotas customizadas identificou-se uma lacuna, uma vez que a maioria das aplicações não consideram as características de cada usuário para a geração de rotas. O INavigS, por sua vez, permite a geração de rotas para o público em geral e para cadeirantes. Em consonância com os quesitos anteriores, o nível de customização foi ajustado de forma a facilitar a construção do modelo espacial e, por conseguinte, a implantação em novos espaços.

No que se refere à assistência durante a rota, outra funcionalidade extremamente desejável, poucos foram os trabalhos que a implementaram. É um tema que carece de estudos em relação à eficiência e à efetividade. Nesse quesito, o INavigS fornece intruções direcionais sob medida e, além disso, informações que permitem a construção de mecanismos mais sofisticados (por exemplo, confirmação da direção a ser seguida).

Os desafios relacionados com o gerenciamento de energia e a disponibilização de APIs não foram abordados por nenhum dos trabalhos. Em relação ao gerenciamento de energia, o INavigS aproveitou-se de diversos mecanismo para salvaguardar energia oferecidos pelo *middleware* M-Hub, bem como, implementou protocolo de comunicação com o servidor (*Websocket*) para que a troca de mensagens consumisse menos energia. Em relação à disponiblização de APIs, viabiliza o uso dos serviços a partir de biblioteca.

Diante do exposto, o INavigS, apesar de não ter se destacado em todos os aspectos de comparação, superou todos os desafios anteriormente elencados. Dessa forma, reconhece-se o INavigS como uma infraestutura de *software* capaz de auxiliar no desenvolvimento de novas aplicações que façam uso de serviços de navegação *indoor*.

5 Conclusão e Trabalhos Futuros

O crescente interesse em aplicações ubíquas, mais precisamente de aplicações baseadas em localização, bem como, as dificuldades que devem ser superadas para o desenvolvimento destas, justificam o surgimento de inúmeras pesquisas na área. Grande parte dessas pesquisas visam, justamente, reduzir o esforço necessário para o desenvolvimento de tais aplicações.

Este trabalho apresentou o INavigS, uma infraestrutura de *software* que visa apoiar o desenvolvimento de novas aplicações, provendo serviços de navegação *indoor*. Aproveitando-se de *beacons* BLE espalhados pelo ambiente, de componentes provenientes de outros projetos e de alguns desenvolvidos durante esta pesquisa, serviços como geocodificação, cálculo de rota, informações acerca do espaço interno e assistência durante a rota são expostos por meio de API.

As principais contribuições deste trabalho são:

- A investigação do estado da arte em relação às ferramentas que utilizaram a tecnologia BLE para prover serviços de navegação *indoor*.
- A proposição de uma arquitetura sensível ao contexto que visa permitir a construção de aplicações ubíquas que façam uso de serviços de navegação *indoor*.
- Projeto e implementação da infraestrutura de *software* INavigS, baseada na arquitetura proposta.

Durante o avanço deste trabalho, foram identificadas diversas oportunidades para extensão, dentre as quais destacam-se:

- Adição de novas tecnologias e métodos, permitindo a navegação em ambientes paramentados com outros tipos de dipositivos, diferentes de *beacons* BLE, ou em ambientes sem qualquer infraestrutura.
- Extensão do componente *Context manager*, adicionando suporte a outros tipos de informações contextuais.

- Extensão da arquitetura para o ambiente *outdoor*, permitindo que sejam construídas aplicações que façam o uso dos serviços de navegação de forma contínua.

- Extensão da customização de rotas, adicionando outros parâmetros ou critérios que permitam gerar rotas cada vez mais ajustadas aos usuários.

- Consideração de aspectos dinâmicos durante a assistência em rota, levando em conta mudanças repentinas como a obstrução de trechos por concentração de pessoas, por exemplo.

Referências Bibliográficas

[1] WEISER, M. The computer for the 21st century. *Scientific american*, Nature Publishing Group, v. 265, n. 3, p. 94–104, 1991.

[2] ABOWD, G. D.; MYNATT, E. D. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, ACM, New York, NY, USA, v. 7, n. 1, p. 29–58, mar. 2000. ISSN 1073-0516. Disponível em: <http://doi.acm.org/10.1145/344949.344988>.

[3] KLEPEIS, N. E. et al. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal of Exposure Science and Environmental Epidemiology*, Nature Publishing Group, v. 11, n. 3, p. 231–252, 2001.

[4] GU, Y.; LO, A.; NIEMEGEERS, I. A survey of indoor positioning systems for wireless personal networks. *IEEE Communications Surveys Tutorials*, v. 11, n. 1, p. 13–32, First 2009. ISSN 1553-877X.

[5] MAINETTI, L.; PATRONO, L.; SERGI, I. A survey on indoor positioning systems. In: *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. [S.l.: s.n.], 2014. p. 111–120.

[6] KALBANDHE, A. A.; PATIL, S. C. Indoor positioning system using bluetooth low energy. In: *2016 International Conference on Computing, Analytics and Security Trends (CAST)*. [S.l.: s.n.], 2016. p. 451–455.

[7] FARD, H. K.; CHEN, Y.; SON, K. K. Indoor positioning of mobile devices with agile ibeacon deployment. In: *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*. [S.l.: s.n.], 2015. p. 275–279. ISSN 0840-7789.

[8] WU, X. et al. ibill: Using ibeacon and inertial sensors for accurate indoor localization in large open areas. *IEEE Access*, v. 5, p. 14589–14599, 2017. ISSN 2169-3536.

[9] ZAFARI, F.; PAPAPANAGIOTOU, I. Enhancing ibeacon based micro-location with particle filtering. In: *2015 IEEE Global Communications Conference (GLOBECOM)*. [S.l.: s.n.], 2015. p. 1–7.

[10] APPLE INC. *Proximity Beacon Specification, Release R1*. [S.l.], 2015. Disponível em `https://developer.apple.com/ibeacon/`. Acessado em 24/06/2018. Disponível em: <https://developer.apple.com/ibeacon/>.

[11] HUANG, H. et al. Smart environment for ubiquitous indoor navigation. In: *2009 International Conference on New Trends in Information and Service Science*. [S.l.: s.n.], 2009. p. 176–180.

[12] SALBER, D.; DEY, A. K.; ABOWD, G. D. The context toolkit: Aiding the development of context-enabled applications. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1999. (CHI '99), p. 434–441. ISBN 0-201-48559-1. Disponível em: <http://doi.acm.org/10.1145/302979.303126>.

[13] GU, T.; PUNG, H. K.; ZHANG, D. Q. A middleware for building context-aware mobile services. In: *2004 IEEE 59th Vehicular Technology Conference. VTC 2004-Spring (IEEE Cat. No.04CH37514)*. [S.l.: s.n.], 2004. v. 5, p. 2656–2660 Vol.5. ISSN 1550-2252.

[14] ROMAN, M. et al. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, v. 1, n. 4, p. 74–83, Oct 2002. ISSN 1536-1268.

[15] KORPIPAA, P. et al. Managing context information in mobile devices. *IEEE Pervasive Computing*, v. 2, n. 3, p. 42–51, July 2003. ISSN 1536-1268.

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

[17] CHEN, G.; KOTZ, D. *A Survey of Context-Aware Mobile Computing Research*. Hanover, NH, USA, 2000.

[18] BASIRI, A. et al. Indoor location based services challenges, requirements and usability of current solutions. *Computer Science Review*, v. 24, p. 1 – 12, 2017. ISSN 1574-0137. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1574013716301782>.

[19] HO, T.-W. et al. Indoor navigation and physician-patient communication in emergency department. In: *Proceedings of the 3rd International Conference on Communication and Information Processing*. New York, NY, USA: ACM, 2017. (ICCIP '17), p. 92–98. ISBN 978-1-4503-5365-6. Disponível em: <http://doi.acm.org/10.1145/3162957.3162971>.

[20] KHALIFA, I. H.; KAMEL, A. E.; BARFETY, B. Real time indoor intelligent navigation system inside hypermarkets. *IFAC Proceedings Volumes*, v. 43, n. 8, p. 461 – 466, 2010. ISSN 1474-6670. 12th IFAC Symposium on Large Scale Systems: Theory and Applications. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1474667015334352>.

[21] IPIÑA, D. López-de; LORIDO, T.; LÓPEZ, U. Indoor navigation and product recognition for blind people assisted shopping. In: BRAVO, J.; HERVÁS, R.; VILLARREAL, V. (Ed.). *Ambient Assisted Living*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 33–40. ISBN 978-3-642-21303-8.

[22] RUBINO, I. et al. Musa: Using indoor positioning and navigation to enhance cultural experiences in a museum. *Sensors*, v. 13, n. 12, p. 17445–17471, 2013. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/13/12/17445>.

[23] BAHL, P.; PADMANABHAN, V. N. Radar: an in-building rf-based user location and tracking system. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. [S.l.: s.n.], 2000. v. 2, p. 775–784 vol.2. ISSN 0743-166X.

[24] SCHILIT, B. N.; THEIMER, M. M. Disseminating active map information to mobile hosts. *IEEE Network*, v. 8, n. 5, p. 22–32, Sept 1994. ISSN 0890-8044.

[25] ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. London, UK, UK: Springer-Verlag, 1999. (HUC '99), p. 304–307. ISBN 3-540-66550-1. Disponível em: <http://dl.acm.org/citation.cfm?id=647985.743843>.

[26] SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*.

Washington, DC, USA: IEEE Computer Society, 1994. (WMCSA '94), p. 85–90. ISBN 978-0-7695-3451-0. Disponível em: <http://dx.doi.org/10.1109/WMCSA.1994.16>.

[27] JANG, S.; KO, E.-J.; WOO, W. Unified user-centric context: Who, where, when, what, how and why. *Personalized Context Modeling and Management for UbiComp Applications*, CEUR-WS, v. 149, n. 26-34, 2005.

[28] CHEN, H. *An intelligent broker architecture for pervasive context-aware systems*. Tese (Doutorado) — University of Maryland, Baltimore County, 2004.

[29] SANTOS, V. V. d. *CEManTIKA: a Domain-independent framework for designing context sensitive systems*. Tese (Doutorado), 2008.

[30] DEY, A. K. *Providing Architectural Support for Building Context-aware Applications*. Tese (Doutorado), Atlanta, GA, USA, 2000. AAI9994400.

[31] GREENBERG, S. Context as a dynamic construct. *Hum.-Comput. Interact.*, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, v. 16, n. 2, p. 257–268, dez. 2001. ISSN 0737-0024.

[32] STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7, 2004*. [s.n.], 2004. LIDO-Berichtsjahr=2004, Potential plagiarism at http://dl.comsoc.org/cocoon/comsoc/servlets/GetPublication?id=9014634. Disponível em: <http://elib.dlr.de/7444/>.

[33] RAYCHOUDHURY, V. et al. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, v. 9, n. 2, p. 177 – 200, 2013. ISSN 1574-1192. Special Section: Mobile Interactions with the Real World. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1574119212001113>.

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

[35] HENRICKSEN, K.; INDULSKA, J. Modelling and using imperfect context information. In: *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*. [S.l.: s.n.], 2004. p. 33–37.

[36] HENRICKSEN, K.; INDULSKA, J.; RAKOTONIRAINY, A. Modeling context information in pervasive computing systems. In: *Proceedings of the First International Conference on Pervasive Computing*. London, UK, UK: Springer-Verlag, 2002. (Pervasive '02), p. 167–180. ISBN 3-540-44060-7. Disponível em: <http://dl.acm.org/citation.cfm?id=646867.706693>.

[37] VIEIRA, V. et al. Uso e representação de contexto em sistemas computacionais. In: *Tópicos em Sistemas Interativos e Colaborativos*. São Carlos, São Paulo, Brasil: UFSCAR, 2006. p. 127–166.

[38] BELLOTTI, V.; EDWARDS, K. Intelligibility and accountability: Human considerations in context-aware systems. *Hum.-Comput. Interact.*, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, v. 16, n. 2, p. 193–212, dez. 2001. ISSN 0737-0024.

[39] TALAVERA, L. E. et al. The mobile hub concept: Enabling applications for the internet of mobile things. In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. [S.l.: s.n.], 2015. p. 123–128.

[40] DAVID, L. et al. A dds-based middleware for scalable tracking, communication and collaboration of mobile nodes. *Journal of Internet Services and Applications*, v. 4, n. 1, p. 16, 2013. ISSN 1869-0238. Disponível em: <http://dx.doi.org/10.1186/1869-0238-4-16>.

[41] VASCONCELOS, R. O.; SILVA, L. D. N. e; ENDLER, M. Towards efficient group management and communication for large-scale mobile applications. In: *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*. [S.l.: s.n.], 2014. p. 551–556.

[42] GOMES, B. d. T. P. et al. A comprehensive and scalable middleware for ambient assisted living based on cloud computing and internet of things. *Concurrency and Computation: Practice and Experience*, v. 29, n. 11, p. e4043, 2017. E4043 cpe.4043. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4043>.

[43] OMG. *Data Distribution Service (DDS) - Version 1.4*. [S.l.], 2015. Disponível em: <http://www.omg.org/spec/DDS/1.4/PDF>.

[44] OMG. *The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification - Version 2.2*. [S.l.], 2014. Disponível em: <http://www.omg.org/spec/DDSI-RTPS/2.2/PDF/>.

[45] BAPTISTA, G. B. et al. A middleware for data - centric and dynamic distributed complex event processing for iot real - time analytics in the cloud. In: . Salvador, Brasil: [s.n.], 2016. (XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC2016).

[46] DAVID, L. et al. A communication middleware for scalable real-time mobile collaboration. In: *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. [S.l.: s.n.], 2012. p. 54–59. ISSN 1524-4547.

[47] EGGUM, M. *Smartphone assisted complex event processing*. Dissertação (Mestrado), 2014.

[48] CORREA, A. et al. A review of pedestrian indoor positioning systems for mass market applications. *Sensors*, v. 17, n. 8, 2017. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/17/8/1927>.

[49] FARID, Z.; NORDIN, R.; ISMAIL, M. Recent advances in wireless indoor localization techniques and system. *Journal of Computer Networks and Communications*, Hindawi Publishing Corporation, v. 2013, 2013. ISSN 2090-7141.

[50] LIU, H. et al. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 37, n. 6, p. 1067–1080, Nov 2007. ISSN 1094-6977.

[51] GOMEZ, C.; OLLER, J.; PARADELLS, J. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, v. 12, n. 9, p. 11734–11753, 2012. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/12/9/11734>.

[52] BLUETOOTH SPECIAL INTEREST GROUP (SIG). *Specification of the Bluetooth® System, Covered Core Package, Version: 4.2*. [S.l.], 2014.

Disponível em https://www.bluetooth.com/specifications/bluetooth-core-specification. Acessado em 26/10/2017. Disponível em: <https://www.bluetooth.com/specifications/bluetooth-core-specification>.

[53] VARSAMOU, M.; ANTONAKOPOULOS, T. A bluetooth smart analyzer in ibeacon networks. In: *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*. [S.l.: s.n.], 2014. p. 288–292. ISSN 2166-6814.

[54] CONTE, G. et al. Bluesentinel: A first approach using ibeacon for an energy efficient occupancy detection system. In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. New York, NY, USA: ACM, 2014. (BuildSys '14), p. 11–19. ISBN 978-1-4503-3144-9. Disponível em: <http://doi.acm.org/10.1145/2676061.2674078>.

[55] APPLE INC. *Getting Started with iBeacon, Version 1.0*. [S.l.], 2014. Disponível em https://developer.apple.com/ibeacon/. Acessado em 24/06/2018. Disponível em: <https://developer.apple.com/ibeacon/>.

[56] KOüHNE, M.; SIECK, J. Location-based services with ibeacon technology. In: *2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation*. [S.l.: s.n.], 2014. p. 315–321.

[57] ESPERTTECH INC. *Esper Reference - Version 4.10.0*. [S.l.], 2013. Disponível em http://esper.espertech.com/release-4.10.0/esper-reference/html_single/index.html. Acessado em 29/07/2018.

[58] KARNEY, C. F. F. Algorithms for geodesics. *Journal of Geodesy*, v. 87, n. 1, p. 43–55, Jan 2013. ISSN 1432-1394. Disponível em: <https://doi.org/10.1007/s00190-012-0578-z>.

[59] RAZINA, E.; JANZEN, D. Effects of dependency injection on maintainability. In: *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*. Anaheim, CA, USA: ACTA Press, 2007. (SEA '07), p. 7–12. ISBN 978-0-88986-706-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1647636.1647639>.

[60] Mozilla Developer Network (MDN) Web Docs. *Promise*. [S.l.]. Disponível em `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise`. Acessado em 29/07/2018.

[61] CHENG, R.-S. et al. Seamless guidance system combining gps, ble beacon, and nfc technologies. *Mobile Information Systems*, v. 2016, n. 2016, 2016.

[62] IVANOV, R. Rsnavi: An rfid-based context-aware indoor navigation system for the blind. In: *Proceedings of the 13th International Conference on Computer Systems and Technologies*. New York, NY, USA: ACM, 2012. (CompSysTech '12), p. 313–320. ISBN 978-1-4503-1193-9. Disponível em: <http://doi.acm.org/10.1145/2383276.2383322>.

[63] PEREIRA, C.; SOUSA, A.; FILIPE, V. Open-source indoor navigation system adapted to users with motor disabilities. *Procedia Computer Science*, v. 67, p. 38 – 47, 2015. ISSN 1877-0509. Proceedings of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050915030938>.

[64] TSIRMPAS, C. et al. An indoor navigation system for visually impaired and elderly people based on radio frequency identification (rfid). *Information Sciences*, v. 320, p. 288 – 305, 2015. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025514008007>.

[65] CHERAGHI, S. A.; NAMBOODIRI, V.; WALKER, L. Guidebeacon: Beacon-based indoor wayfinding for the blind, visually impaired, and disoriented. In: *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. [S.l.: s.n.], 2017. p. 121–130.

[66] CHANDEL, V. et al. Inloc: An end-to-end robust indoor localization and routing solution using mobile phones and ble beacons. In: *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. [S.l.: s.n.], 2016. p. 1–8.

[67] LI, F. et al. A reliable and accurate indoor localization method using phone inertial sensors. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. New York, NY, USA: ACM, 2012. (UbiComp '12), p. 421–430. ISBN 978-1-4503-1224-0. Disponível em: <http://doi.acm.org/10.1145/2370216.2370280>.

[68] ZHOU, Y. An efficient least-squares trilateration algorithm for mobile robot localization. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2009. p. 3474–3479. ISSN 2153-0858.

[69] AHMETOVIC, D. et al. Navcog: A navigational cognitive assistant for the blind. In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*. New York, NY, USA: ACM, 2016. (MobileHCI '16), p. 90–99. ISBN 978-1-4503-4408-1. Disponível em: <http://doi.acm.org/10.1145/2935334.2935361>.

[70] AHMETOVIC, D. et al. Achieving practical and accurate indoor navigation for people with visual impairments. In: *Proceedings of the 14th Web for All Conference on The Future of Accessible Work*. New York, NY, USA: ACM, 2017. (W4A '17), p. 31:1–31:10. ISBN 978-1-4503-4900-0. Disponível em: <http://doi.acm.org/10.1145/3058555.3058560>.

[71] KIM, J.-E. et al. Navigating visually impaired travelers in a large train station using smartphone and bluetooth low energy. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2016. (SAC '16), p. 604–611. ISBN 978-1-4503-3739-7. Disponível em: <http://doi.acm.org/10.1145/2851613.2851716>.

[72] BLATTNER, A.; VASILEV, Y.; HARRIEHAUSEN-MüHLBAUER, B. Mobile indoor navigation assistance for mobility impaired people. *Procedia Manufacturing*, v. 3, p. 51 – 58, 2015. ISSN 2351-9789. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2351978915001080>.

[73] HARRIEHAUSEN-MUHLBAUER, B. WheelScout - Mobile outdoor and indoor navigation via voice control for limited mobility users. *IMSCI 2016 - 10th International Multi-Conference on Society, Cybernetics and Informatics, Proceedings*, n. Imsci, p. 212–217, 2016.

[74] HERWIG, V.; FISCHER, R.; BRAUN, P. Assessment of rest and websocket in regards to their energy consumption for mobile applications. In: *2015 IEEE*

8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). [S.l.: s.n.], 2015. v. 1, p. 342–347.

A Diagramas de sequência

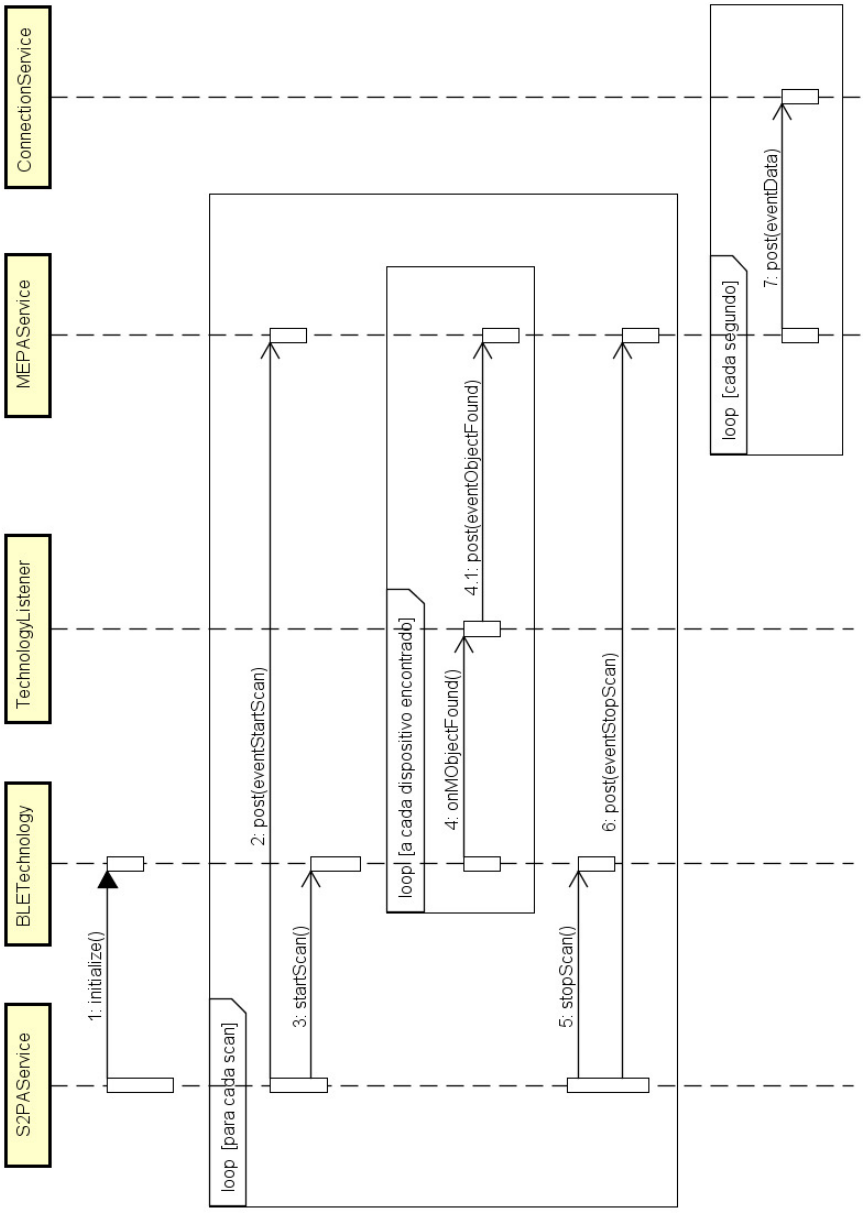

Figura A.1: Diagrama de sequência do processo de descoberta.

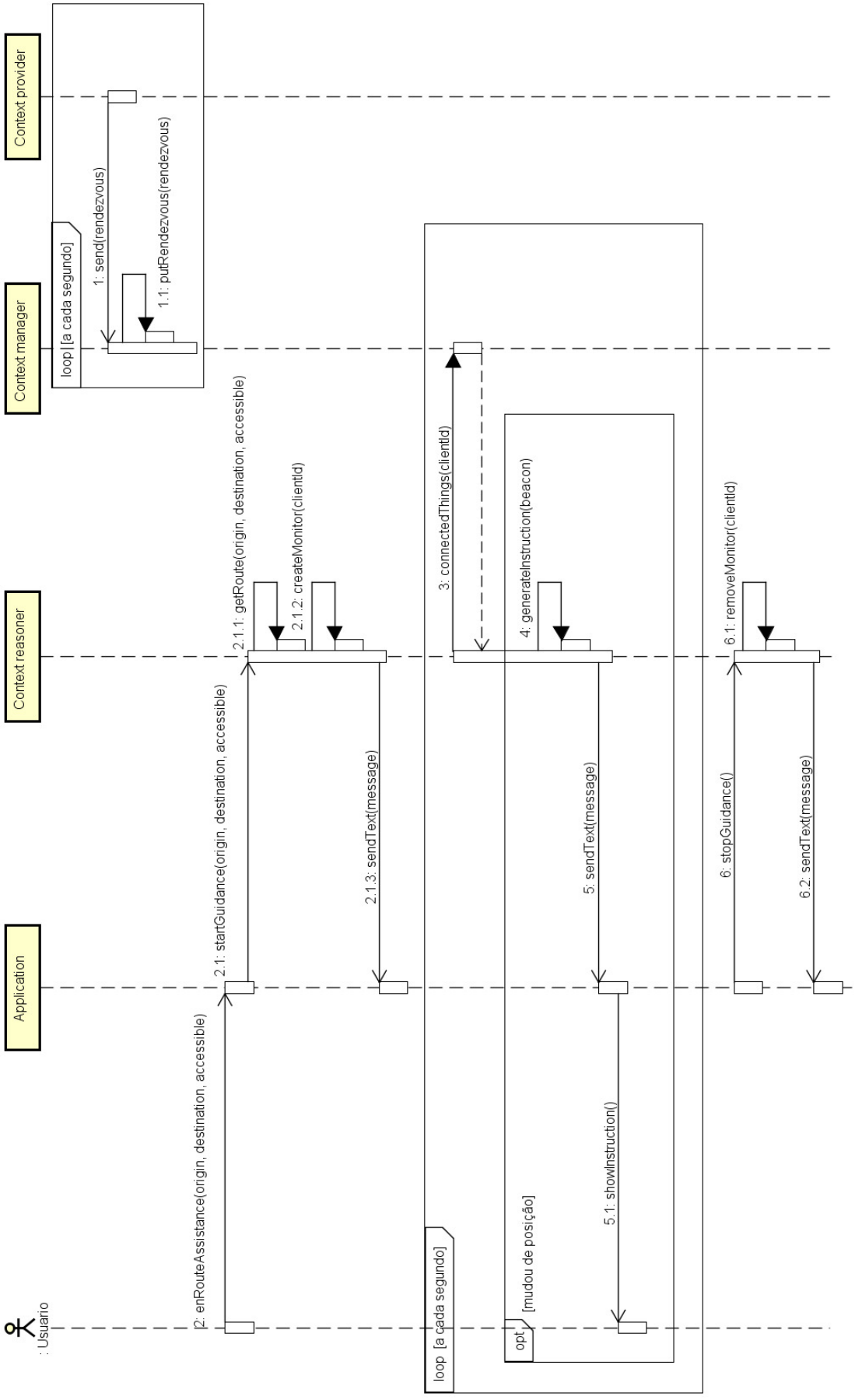

Figura A.2: Diagrama de sequência do serviço de assistência durante a rota.

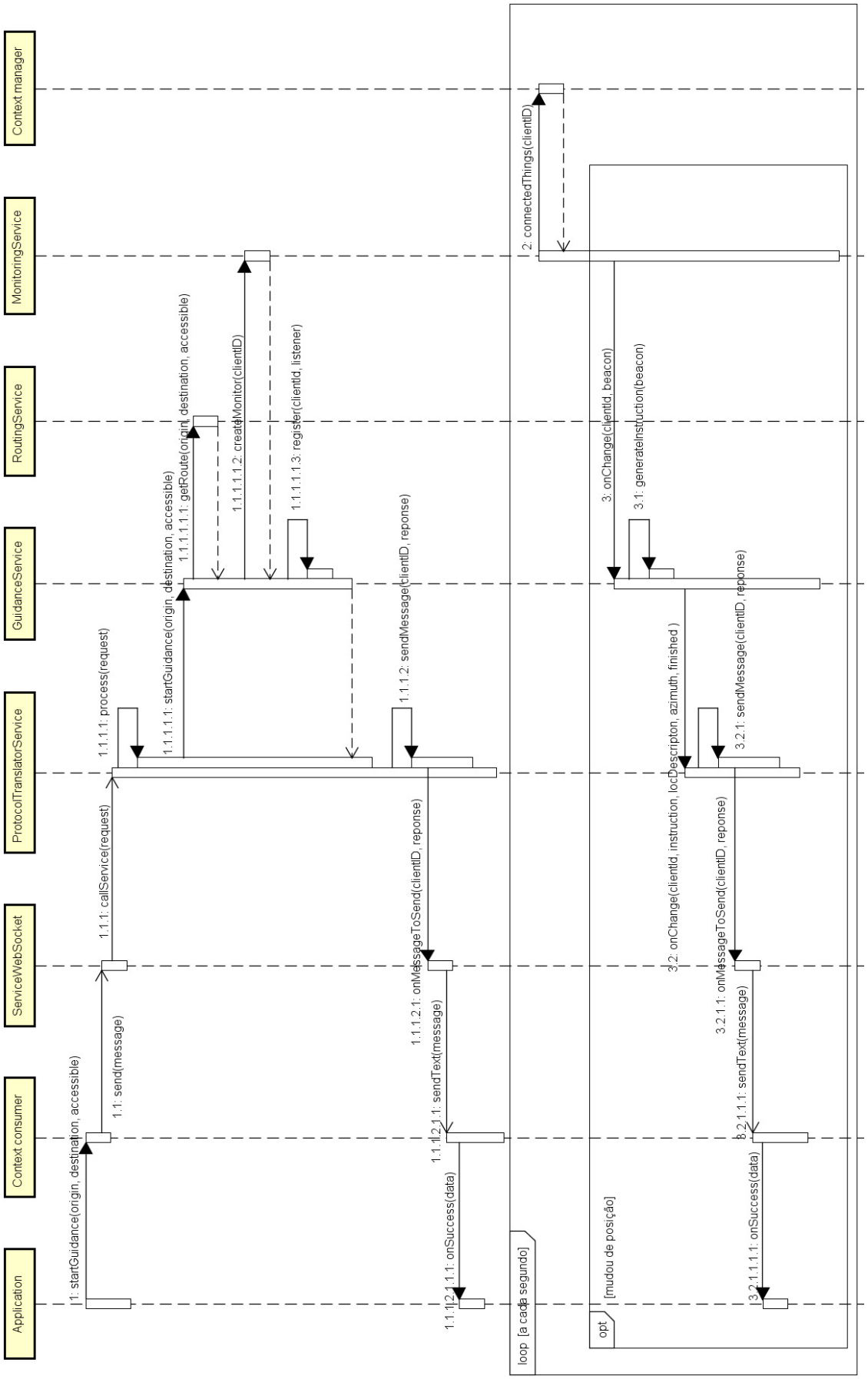

Figura A.3: Diagrama de sequência do serviço de assistência durante a rota (*push-based*), detalhando interação dos subcomponentes do *Context reasoner*.

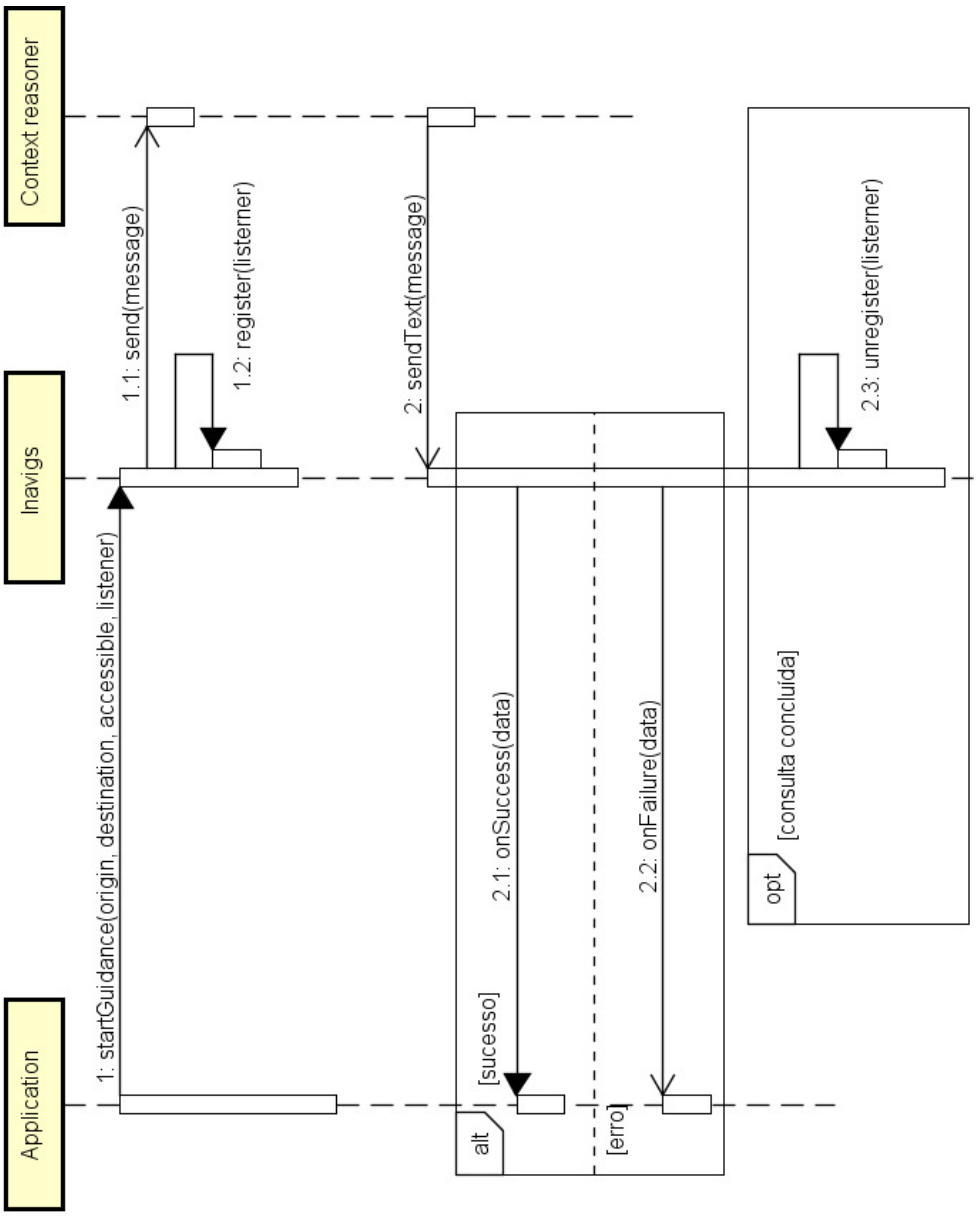

Figura A.4: Diagrama de sequência do serviço de assistência durante a rota sob a perspectiva do *Context consumer*.

B Listagens

Listagem B.1: Código EPL para criação da *context partition* scanBLE.

```
CREATE CONTEXT scanBLE
        START TechnologyData(idTechnology=BLETechnology.ID,
            action=TechnologyData.START_SCAN)
        END TechnologyData(idTechnology=BLETechnology.ID,
            action=TechnologyData.STOP_SCAN)
```

Listagem B.2: Código EPL para inserção do evento com maior RSSI no fluxo de eventos BeaconData.

```
CONTEXT scanBLE
        INSERT INTO BeaconData
            SELECT * FROM SensorData
                OUTPUT all when terminated
                ORDER BY signal DESC LIMIT 1
```

Listagem B.3: Código EPL para consultar o último encontro processado.

```
SELECT * FROM BeaconData.win:length(1)
    OUTPUT snapshot every 1 second
```

Listagem B.4: Descrição do protocolo HORYS API versão 0.1.

```
HORYS API Protocol 0.1
    mode: 1 (request) or 2 (reply);
    operation:
        0 = PUTRENDEZVOUS;
        1 = GETCONNECTEDTHINGS;
        2 = GETCONNECTEDMHUBS;
        3 = GETAVGRENDEZVOUSDURATION;
    parameters: set of k,v pairs { key : value }
```

Listagem B.5: JSON em resposta à consulta *Connected M-Hubs* do HORYS.

```
{   "response":["ec52c1bd-2982-e861-2f70-d07d8a9197a0"],
    "operation":"GETCONNECTEDMHUBS",
    "thingID":"acaeed86-b05e-c9ce-a553-fa3303d7101f" }
```

Listagem B.6: JSON em resposta à consulta *Connected Things* do HORYS.

```
{   "response": ["2f234454-cf6d-4a0f-adf2-f4911ba9ffa6",
                    "b1b7d88b-24f5-4bf5-912d-eb429133b1d4"],
    "operation":"GETCONNECTEDTHINGS",
    "mhubID":"1c9ce246-6276-47b9-86c4-8b2aba82e6ce"}
```

Listagem B.7: Exemplo de código em Java para configuração e inicialização da API.

```
1  //definindo propriedades
2  Properties properties = new Properties();
3  properties.put(Inavigs.PROPERTY_HOST, "192.168.100.1");
4  properties.put(Inavigs.PROPERTY_PORT, "8081");
5  properties.put(Inavigs.PROPERTY_VERSION, "v1");
6  properties.put(Inavigs.PROPERTY_DEBUG, "false");
7  properties.put(Inavigs.PROPERTY_PROTOCOL, Inavigs.WEBSOCKET_PROTOCOL);
8
9  //configurando e inicializando a biblioteca
10 Inavigs.getInstance().init(properties).start(clientUUID,
11     new ServiceConsumerStrategy.ServiceListener() {
12
13         //metodo executado em caso de sucesso
14         @Override
15         public void onSuccess(Map<String, Object> map) {...};
16
17         //metodo executado em caso de erro
18         @Override
19         public void onFailure(Integer statusCode, String message) {...}
20     });
```

Listagem B.8: Exemplo de código em Javascript para configuração e inicialização da API.

```
//veriricando se a biblioteca foi carregada
if (!window.inavigs) {
    console.log("Inavigs not set.");
}else{

    //configurando biblioteca
    inavigs.setOptions({
        protocol: "websocket", // websocket or http
        host: "localhost",
        port: "8081",
        version: "v1",
        debug:false
    });

    //inicializando biblioteca
    inavigs.start(cUUID)
        //funcao executada em caso de sucesso
        .then((event)=>{...})
        //funcao executada em caso de erro
        .catch((error)=>{...});
}
```

Listagem B.9: Exemplo de código em Java para inicialização do serviço de assistência durante a rota.

```
//listener para o servico de assistencia durante a rota
private final ServiceConsumerStrategy.ServiceListener slGuidance =
        new ServiceConsumerStrategy.ServiceListener() {
            //metodo executado em caso de sucesso
            //sera executado ao receber instrucoes
            @Override
            public void onSucess(Map<String, Object> map) {
                //verificando se a colecao possui itens
                if (map != null && !map.isEmpty()) {

                    //instrucao recebida
                    final String instruction =
                        map.get("instruction").toString();
                }
```

```
15              }
16
17              //metodo executado em caso de erro
18              @Override
19              public void onFailure(Integer statusCode, String message) {...}
20          };
21
22
23  public void startGuidance() {
24      //iniciando o servico de assistencia durante a rota
25      Inavigs.getInstance().startGuidance(origin, destination, accessible,
26          slGuidance);
27  }
```

Listagem B.10: Exemplo de código em Javascript para inicialização do serviço de assistência durante a rota.

```
1  //funcao executada em caso de sucesso
2  function onSucess(event){
3      //verificando a nulidade do parametro
4      if(event.detail !== undefined && event.detail !== null
5              && event.detail.data.instruction !== undefined
6              && event.detail.data.instruction !== null ){
7          //escrevendo a instrucao recebida no console
8          console.log("Instrucao: " + event.detail.data.instruction);
9      }
10 }
11
12 //funcao executada em caso de erro
13 function onError(event){
14     console.log("Error: " + event.detail.data.message);
15 }
16
17 function startGuidance(){
18     //iniciando o servico de assistencia durante a rota
19     inavigs.startGuidance(
20             $('#sOrigin').val(),
21             $('#sDestination').val(),
22             false, onSucess, onError).then((event)=>{
```

```
23                                  console.log("..");
24                                }).catch(onError);
25 }
```