Chapter 1

Computação Móvel e Computação Ubíqua: Visão Geral e Exemplo de Uso

Marcio Pereira de Sá

Resumo

A Computação Móvel é a área da Computação responsável pelas pesquisas relacionadas ao desenvolvimento tanto de dispositivos móveis (hardware), quanto de aplicações e sistemas que são executados nesses dispositivos (software). Por outro lado, a Computação Ubíqua, se refere ao uso coletivo de computadores disponíveis no ambiente físico dos usuários, talvez embutidos em uma forma invisível para esses usuários. À medida em que novos dispositivos móveis são disponibilizados e novas tecnologias de sensoriamento são desenvolvidas, um novo tipo de aplicações computacionais começam a se popularizar; elas são denominadas aplicações sensíveis ao contexto e se caracterizam por perceber e reagir às mudanças no ambiente que as envolve. Porém, para superar os desafios para desenvolvê-las, há muitas pesquisas a fim de construir infraestruturas que facilitem a criação e a execução de aplicações sensíveis ao contexto e o presente trabalho procura ilustrar as caracterísiticas principais dessas infraestruturas.

1.1. Introdução

Computação Móvel é a área da Computação responsável pelas pesquisas relacionadas ao desenvolvimento tanto de dispositivos móveis (*hardware*), tais como aparelhos telefônicos celulares, *smartphones*, PDAs, livros digitais e navegadores GPS, quanto de aplicações e sistemas que são executados nesses dispositivos (*software*). É atualmente uma das áreas da Tecnologia da Informação mais importantes, pois tem experimentado, nas últimas duas décadas, um crescimento espetacular a ponto de alguns considerarem a Computação Móvel como a quarta revolução da Computação [1], antecedida pelos grandes centros de processamento de dados da década de 1960, o surgimento dos terminais nos anos 1970 e as redes de computadores e a popularização dos computadores pessoais na década de 1980.

Por outro lado, a Computação Ubíqua, através da definição dada por Mark Weiser [2], se refere ao uso coletivo de computadores disponíveis no ambiente físico dos usuários,

talvez embutidos em uma forma invisível para esses usuários. É a computação pervasiva, onde os sistemas computacionais estão à disposição de seus usuários praticamente em qualquer lugar e a qualquer momento.

É a união dessas duas importantes áreas da Computação (Computação Móvel e Computação Ubíqua) que permite o desenvolvimento de programas de computador ou aplicações computacionais mais úteis e acessíveis a seus usuários. Dentre essas aplicações móveis e ubíquas mais conhecidas, estão os *serviços baseados em localização* (LBS - *Location-based Service*). Estas aplicações fornecem informações aos seus usuários de acordo com o local em que se encontram. Como exemplo, poderíamos utilizar um sistema que notificasse seus usuários sempre que estes estivessem próximos a lojas, pontos turísticos ou de lazer de interesse desses usuários. Estão também entre as categorias de aplicações móveis mais populares e interessantes os jogos eletrônicos para *smartphones*, as agendas de compromissos e as redes sociais para dispositivos móveis.

Atualmente, há vários tipos de dispositivos móveis disponíveis no mercado, tais como aparelhos telefônicos celulares comuns, telefones inteligentes (*smartphones*), PDAs (*Personal Digital Assistant* - Assistente Digital Pessoal), *tablets*, rastreadores com receptores GPS e leitores de livros eletrônicos (*e-readers*). Apesar da diversidade de dispositivos móveis existentes, um grande nicho em especial se abre para os profissionais da Computação Móvel: o mercado de telefones inteligentes ou *smartphones*. Esses aparelhos possuem sistemas operacionais próprios e recursos de hardware (telas sensíveis ao toque, acelerômetros, receptores GPS, câmeras fotográficas, dentre outros) que viabilizam o desenvolvimento de aplicações móveis cada vez mais adaptadas ao ambiente e a seus usuários.

1.2. Computação Móvel

Como já foi dito na Seção 1.1, a Computação Móvel se ocupa com as pesquisas e o desenvolvimento tanto de dispositivos móveis (*hardware*), quanto de aplicações e sistemas que são executados nesses dispositivos (*software*). Nesse sentido é importante analisar aspectos tecnológicos tanto de hardware quanto de software. Em relação ao hardware, algumas características são especialmente importantes para o nosso estudo, como as diferentes gerações tecnológicas da telefonia móvel. Aspectos importantes de software incluem as aplicações móveis e suas plataformas de desenvolvimento, como o iPhone e o Android. Nas Seções seguintes estes e outros aspectos serão analisados em maiores detalhes.

1.2.1. Breve Histórico da Computação Móvel

De acordo com [1], Em linhas gerais, a Computação Móvel tem como precursores os sistemas analógicos de comunicação que têm em sua origem remota a descoberta feita por Hans Christian Oersted, em 1820, de que a corrente elétrica produz um campo elétrico. A partir daí foi possível desenvolver vários sistemas de comunicação, cujo primeiro foi o telégrafo. Este sistema, inicialmente baseado na comunicação com fio, ainda na metade do século XIX, permitia a transferência de palavras a longa distância através do código Morse. Em seguida, após a descoberta das equações de Maxwell, que descrevem a propagação de ondas eletromagnéticas, aliada aos experimentos de Heinrich Hertz, foi possível a descoberta da radiotelegrafia por Marconi, no final do século XIX. Em 1901, o Oceano

Atlântico era atravessado por sinais de rádio. Estavam lançadas as bases para a comunicação sem fio.

A segunda geração de sistemas de comunicação desenvolvida foi o telefone, inventado por Alexander Graham Bell. Em seguida, com o advento dos computadores, temos a terceira geração dos sistemas de comunicação, pois, através dos sistemas computacionais, a comutação telefônica também se torna digital, reduzindo de forma drástica a necessidade de operadores humanos no sistema de comutação telefônico. Até este momento, ainda predominava a comunicação com fio, caracterizada pelo elevado custo de acesso remoto. Por causa disso, os sistemas sem fio se tornaram atraentes. Porém, nesta época, ainda dependiam significativamente das redes fixas.

De modo geral, o primeiro sistema de comunicação móvel existente foi um sistema de rádio utilizado pela polícia de Detroit, nos Estados Unidos, em 1928. Já em 1947, a AT&T desenvolve o IMTS (*Improved Mobile Telephone Service*), um sistema de transmissão em que era necessário somente uma única torre de alta potência para atender a uma grande área ou cidade. Posteriormente, nos Anos 1970, a AT&T lança o sistema celular denominado AMPS (Advanced Mobile Phone System). No início, era um sistema utilizado em automóveis e atendiam um pequeno número de usuários com uma capacidade muito limitada de tráfego. O Japão foi o país que recebeu a primeira rede celular no mundo, no final da década de 1970, mais precisamente em 1979. A segunda geração do sistema AMPS aparece em 1983, com a primeira rede celular americana que operava em Chicago e Baltimore.

A década de 1990 foi relativamente produtiva para a telefonia celular. Já em 1991, acontece a validação inicial dos padrões TDMA e CDMA nos Estados Unidos e a introdução da tecnologia microcelular. Um ano mais tarde, em 1992, aparece o sistema Pan-Europeu GSM (Groupe Spéciale Mobile). Em meados dessa década, temos a introdução do sistema CDPD (Cellular Digital Packet Data) e início dos serviços PCS (Personal Communication Services) CDMA e TDMA, em 1994. Os projetos para a cobertura terrestre por satélites de baixa órbita, como o projeto Iridium são iniciado em 1995. A década de 2000 é marcada principalmente pela popularização do 3G e das redes Wi-Fi.

O restante do texto está assim organizado. Na Seção 1.2 será feita uma análise sobre os principais termos e tecnologias empregadas na Computação Móvel. A Seção 1.3, por outro lado, apresentará as bases da Computação Ubíqua e Pervasiva, tratando especialmente das arquiteturas de provisão de contexto. A Seção 1.4 apresenta as considerações finais, procurando evidenciar a importância das pesquisas nestas duas grandes áreas da Computação.

1.2.2. As Geração da Telefonia Móvel

De maneira geral, podemos organizar a história e a evolução da telefonia móve em gerações. Estas gerações são caracterizadas principalmente por suas tecnologias de comunicação. A seguir são listadas as gerações e suas características.

Primeira Geração - 1G. Esta foi a primeira geração de telefonia móvel. Dentre suas características principais, destacam-se o emprego de tecnologia analógica e o uso de aparelhos volumosos em relação aos aparelhos celulares atuais. Os principais

padrões empregados pela primeira geração foram: AMPS (*Advanced Mobile Phone System*), desenvolvido nos Estados Unidos, ainda na década de 1970. É considerado o primeiro padrão de rede celular. Um outro padrão semelhante é o TAC (*Total Access Communication System*), uma versão europeia do padrão AMPS. Temos ainda o ETACS (*Extended Total Access Communication System*), considerado uma evolução do TAC.

Segunda Geração - 2G. A segunda geração de telefonia móvel tem como característica principal a passagem da tecnologia analógica para a digital e, consequentemente, houve também uma diminuição do tamanho e do peso dos aparelhos celulares. Dentre os padrões de segunda geração, destacam-se: GSM *(Global System for Mobile communications)*, é considerado o principal padrão 2G na Europa e utiliza as bandas de frequência de 900 MHz e 1800 MHz (na Europa) e 1900 MHz (nos Estados Unidos). O CDMA (*Code Division Multiple Access*) e o TDMA (*Time Division Multiple Access*) são tecnologias alternativas ao GSM europeu.

Outra característia importante das tecnologias 2G é a capacidade de transmitir além da voz, dados digitais de pouco volume, como mensagens de texto curtas (SMS - *Short Message Service*) ou mensagens multimídia (MMS - *Multimedia Message Service*). O padrão GSM original permite transferência a, no máximo, 9,6 Kbps.

Gerações 2.5G e 2.75G O GPRS (*General Packet Radio System*) foi desenvolvido como uma extensão do padrão GSM, permitindo velocidades de transferência de dados teóricos até 114 Kbps. Por ser uma evolução do padrão GSM de segunda geração, o GPRS recebeu a denominação de 2.5G. Já o padrão EDGE (*Enhanced Data Rates for Global Evolution*) conseguiu elevar a velocidade de transferência teórica para cerca de 384 Kbps, tornando viável o uso de algumas aplicações multimídias. Por isso, o EDGE ficou conhecido como um padrão 2.75G.

Terceira Geração - 3G. Dentre as principais características da terceira geração de telefonia móvel (3G), especificadas pelo IMT-2000 (*International Mobile Telecommunications for the year 2000*) da União Internacional das Comunicações (UIT), destacam-se: altas taxas de transmissão de dados, compatibilidade mundial e compatibilidade dos serviços de terceira geração com os sistemas de segunda geração. O principal padrão 3G é denominado UMTS (*Universal Mobile Telecommunications System*), utilizando uma banda de frequência de 5 MHz, podem alcançar velocidades de transferência de dados de 384 Kbps a 2 Mbps. O padrão UMTS emprega a codificação W-CDMA (*Wideband Code Division Multiple Access*). Uma evolução do padrão 3G é a tecnlogia HSDPA (High-Speed Downlink Packet Access), considerado um padrão 3.5G. O HSDPA permite taxas de transferência entre 8 a 10 Mbps.

1.3. Computação Ubíqua

Imagine que, ao entrar em uma sala de reuniões, na empresa onde trabalha, seu telefone celular automaticamente mudasse o tipo de toque de campainha para *silencioso*, pois o seu

aparelho “inteligente” sabia que você (através de sua agenda de compromissos hospedada na Web), a partir daquele momento até às dez e trinta da manhã, estaria em uma reunião muito importante e não desejaria que fosse incomodado de modo abrupto, evitando assim contrangimentos. Imagine ainda que, no final da tarde ao chegar em casa, o sistema de climatização de sua residência automaticamente acionasse os condicionadores de ar, deixando a residência na sua temperatura preferida. Logo após, a campainha tocasse com um rapaz entregando-lhe algumas compras feitas também de forma automática por sua geladeira que notou a carência desses produtos.

Este tipo de comportamento, até certo ponto futurista, já não existe apenas na imaginação visionária de alguns cineastas, nem é privilégio de alguns pesquisadores trabalhando em complexos laboratórios de pesquisa de novas tecnologias. Atualmente, com o avanço da computação móvel e das técnicas de computação ubíqua, isto já é uma realidade.

Porém, desenvolver sistemas computacionais que possam se comportar de modo mais “inteligente” e mais útil aos seus usuários só é possível se tais sistemas forem dotados de capacidades parecidas com aquelas encontradas nos seres vivos, especialmente os seres humanos, como a capacidade de sentir o ambiente a sua volta e reagir às mudanças neste ambiente. Sistemas computacionais dotados dessas características são, então, denominados *Sistemas ou Aplicações Sensíveis ao Contexto*, pois usam as mudanças nas informações de contexto, como localização, data e hora, agenda de compromissos, dentre outras para reagir e adaptar seu comportamento de acordo com tais mudanças.

Entretanto, deve-se observar que não apenas os grandes e complexos sistemas computacionais, como casas inteligentes e sistemas de segurança de algumas instituições financeiras, podem se tornar sensíveis ao contexto. O uso de informações de contexto para melhorar a interação com usuários já é empregado há muito tempo em aplicações relativamente simples, como em sistemas de busca pela Web, aplicações para o envio de anúncios inteligentes, sistemas de tradução de textos, etc.

Além disso, o uso da sensibilidade ao contexto oferece muitas oportunidades para a criação de aplicações mais adaptadas ao ambiente móvel e ubíquo, porém há também muitos desafios para desenvolvê-las. Dentre estes desafios estão a grande diversidade de informações contextuais (localização dos usuários, luminosidade ambiente, uso de CPU e memória, qualidade da rede sem fio e muitas outras) e a abundância de tecnologias de sensoriamento, como, por exemplo, vários modelos de sensores de temperatura, cada um com uma interface de acesso própria ou ainda diferentes APIs para acesso a agendas de compromissos de usuários hospedadas na Web. Nota-se também que, em geral, as informações de contexto necessárias para desenvolver estas aplicações podem ser obtidas de muitas formas distintas, como através do uso de sensores físicos para a obtenção de dados sobre a temperatura ou luminosidade do ambiente, informações sobre o status do dispositivo (quantidade de memória livre, nível de uso da CPU, qualidade da rede); ou através ainda de sensores lógicos, fornecendo informações sobre a agenda de compromissos dos usuários, suas preferências, suas páginas Web mais visitadas, dentre outras.

Por isso, desenvolver aplicações sensíveis ao contexto sem nenhuma infraestrutura de provisão de contexto não é tarefa simples, exigindo que o programador dedique grande parte de seu tempo em tarefas relacionadas à coleta, processamento e disponibilização de

informações de contexto. Esse tempo precioso dedicado a essas tarefas poderia ser melhor empregado em atividades mais diretamente relacionadas à lógica da própria aplicação a ser desenvolvida em vez de gastá-lo em tarefas relacionadas à provisão de informações contextuais. Um outro problema desse estilo de programação é a dificuldade em manter a aplicação, pois se um sensor tiver que ser substituído por outro com uma interface de acesso aos dados diferente, o programador deverá modificar boa parte do código de sua aplicação para que a mesma possa agora se comunicar com o novo sensor e receber dele os dados coletados.

Portanto, para facilitar o desenvolvimento de aplicações sensíveis ao contexto e popularizar o uso dessas aplicações junto aos usuários finais, é necessário o uso de infraestruturas para a provisão de contexto (plataformas de *middleware*, *frameworks*, *toolkits*, etc.). Essas infraestruturas fornecem mecanismos que facilitam a coleta, processamento e disponibilização de informações de contexto às aplicações, permitindo que os programadores possam se concentrar, principalmente, em assuntos mais diretamente relacionados à lógica específica de sua aplicação, deixando assuntos inerentes à provisão de contexto, como a comunicação adequada com todos os sensores, o processamento dessas informações e a sua disponibilização sob a responsabilidade dessas plataformas de provisão de contexto.

Esta Seção tem como objetivos principais introduzir os principais conceitos sobre a Computação Sensível ao Contexto, bem como analisar algumas plataformas de provisão de contexto, procurando evidenciar suas particularidades, qualidades, contribuições e limitações. Finalmente, discutir os principais desafios enfrentados pelos desenvolvedores de aplicações sensíveis ao contexto.

1.3.1. Classificação, Histórico e Evolução da Computação Sensível ao Contexto

De acordo com o exposto anteriormente, os sistemas computacionais que utilizam informações contextuais para reagir e se adaptar a mudanças no ambiente que o circunda são denominados sistemas ou aplicações sensíveis ao contexto e, por conseguinte, a subárea da Computação que estuda e desenvolve tais sistemas é denominada Computação Sensível ao Contexto. Deve-se ressaltar, ainda, que a Computação Sensível ao Contexto é, na verdade, uma subárea dentro de uma área mais ampla denominada Computação Ubíqua ou Pervasiva que, por sua vez, está situada na subárea da Computação Geral denominada Redes de Computadores e Sistemas Distribuídos. Além disso, a união da Computação Sensível ao Contexto com a Computação Móvel dá origem a uma nova e promissora área, conhecida por Computação *Móvel* Sensível ao Contexto. A Figura 1.1 ilustra essa classificação.

Para Mark Weiser [2], a Computação Ubíqua está relacionada ao uso coletivo de computadores disponíveis no ambiente físico dos usuários, podendo até estar embutido em uma forma invisível para estes mesmos usuários. Uma outra definição semelhante é a da Computação Pervasiva que se refere à visão de dispositivos ou computadores fazendo parte efetiva da vida das pessoas. Este fenômeno pode ser visto, por muitos, como uma combinação da Computação Móvel (ou seja, o uso de computadores instalados no próprio corpo dos usuários ou que podem ser carregados por estes) e os computadores embutidos nos ambientes fixos, podendo, por este fato, ser compreendida como sinônimo para a

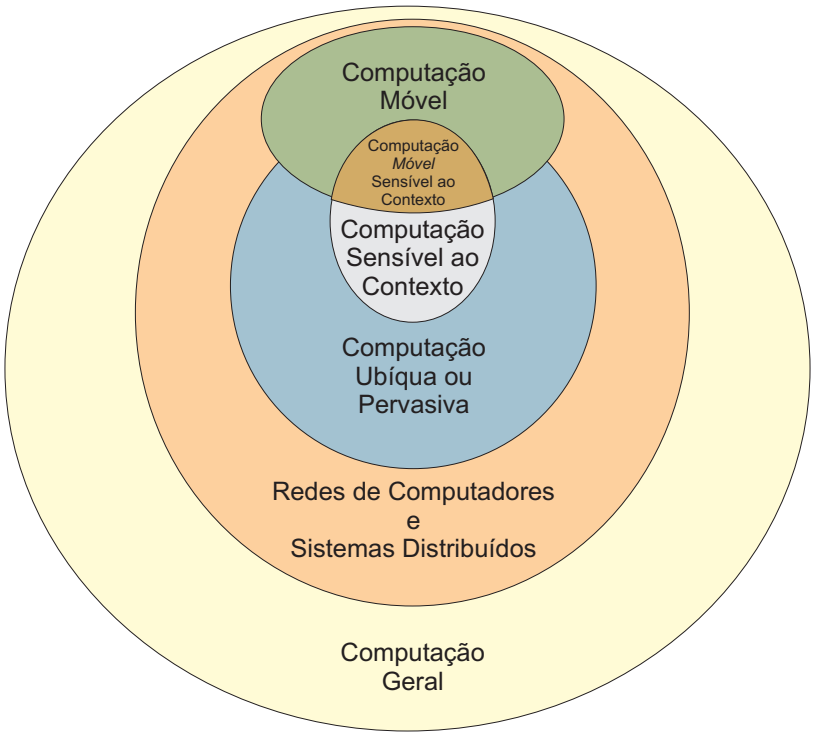

Figure 1.1. O relacionamento entre a Computação Sensível ao Contexto e outras áreas da Computação Geral.

computação ubíqua [3].

Segundo Dey et al [4], o termo "sensibilidade ao contexto" foi usado, pela primeira vez, em 1994, no trabalho dos pesquisadores B. Schilit e M. Theimer [5]. Neste trabalho, os autores se referiam à *sensibilidade ao contexto* como sendo a capacidade que um software possui para se adaptar ao local em que está sendo executado, ao conjunto de pessoas e objetos próximos, e também às mudanças dessas pessoas e desses objetos ao longo do tempo. Porém, antes mesmo dessa publicação, ainda em 1992, um trabalho pioneiro já estava sendo desenvolvido: o Active Badge Location System da Olivetti [6] (um sistema baseado em localização). Este e outros trabalhos contemporâneos são, de alguma forma, derivados da visão visionária de Mark Weiser [2], ainda em 1991, sobre uma nova forma de desenvolver e utilizar software, com capacidades avançadas de interação tanto com os usuários como com o ambiente em que é executado.

Para ilustrar essa pequena história da Computação Sensível ao Contexto e, por conseguinte, da Computação Ubíqua, uma linha do tempo sobre alguns fatos importantes da Computação Sensível ao Contexto é exibida na Figura 1.2.

1.3.2. O que é Contexto

Conforme observado por [3], a noção de contexto tem sido empregada em inúmeras áreas, incluindo linguística, filosofia, representação do conhecimento, teoria da comunicação e resolução de problemas no campo da inteligência artificial. Portanto, é natural encontrarmos várias definições de contexto, de acordo com a área de interesse. De modo geral, por

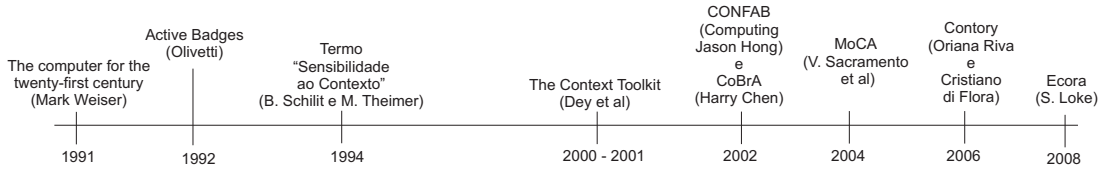

Figure 1.2. Alguns fatos importantes sobre a evolução da Computação Sensível ao Contexto.

exemplo, o *Free Online Dictionary of Computing* [7] define contexto como "aquilo que está em volta, e dá significado a algo." Com certeza esta é uma definição bastante ampla e não nos auxilia muito quando o assunto é especificamente a área de Sistemas Distribuídos e Redes de Computadores.

Nota-se, como mencionado anteriormente, que o trabalho de M. Theimer e B. Schilit, referia-se a contexto como sendo a informação de localização, a identidade das pessoas e objetos colocalizados e ainda as mudanças nesses objetos. Entretanto, ao longo dos anos seguintes, vários pesquisadores têm procurado definir o significado do termo "Contexto" sob o ponto de vista da Computação. Ryan *et al.* [8] definem, de modo similar, o termo contexto como sendo a localização do usuário, o ambiente, a identidade e o tempo.

Dey *et al.* [4] nos dá uma definição mais operacional de contexto, que nos parece ser mais útil na prática e mais adequada à computação ubíqua. Para eles, *contexto é qualquer informação que possa ser usada para caracterizar a situação de entidades (por exemplo, pessoas, locais ou objetos) que são consideradas relevantes para a interação entre um usuário e uma aplicação, incluindo também o usuário e a aplicação*. No restante desse capítulo, esta será a definição que empregaremos ao usar este termo.

A definição de Dey *et al.*, descrita acima, é ilustrada na Figura 1.3

1.3.2.1. Tipos e Categorias de Contexto

De modo semelhante à definição de contexto, durante anos vários pesquisadores procuraram classificar as informações de contexto, segundo alguns critérios e aspectos. A seguir, analisamos algumas dessas classificações.

T. Gu *et al.* [9] admitem dois tipos ou categorias principais de contexto: *contexto direto* e *contexto indireto*.

Para eles, *contexto direto* se refere àquele obtido ou adquirido diretamente de um provedor de contexto que pode ser uma fonte interna, como um provedor de localização interno (indoor) ou uma fonte externa, como um servidor de informações sobre meteorologia. Além disso, subdividem o contexto direto em contexto que pode ser sentido (*sensed context*) e contexto definido (*defined context*). O *sensed context* é adquirido ou obtido por *sensores físicos*, tais como a medida da temperatura de uma sala obtida por um sensor posicionado no teto da mesma, ou por *sensores virtuais*, como um *web service* fornecendo informações sobre a temperatura em uma determinada cidade distante. O *de-*

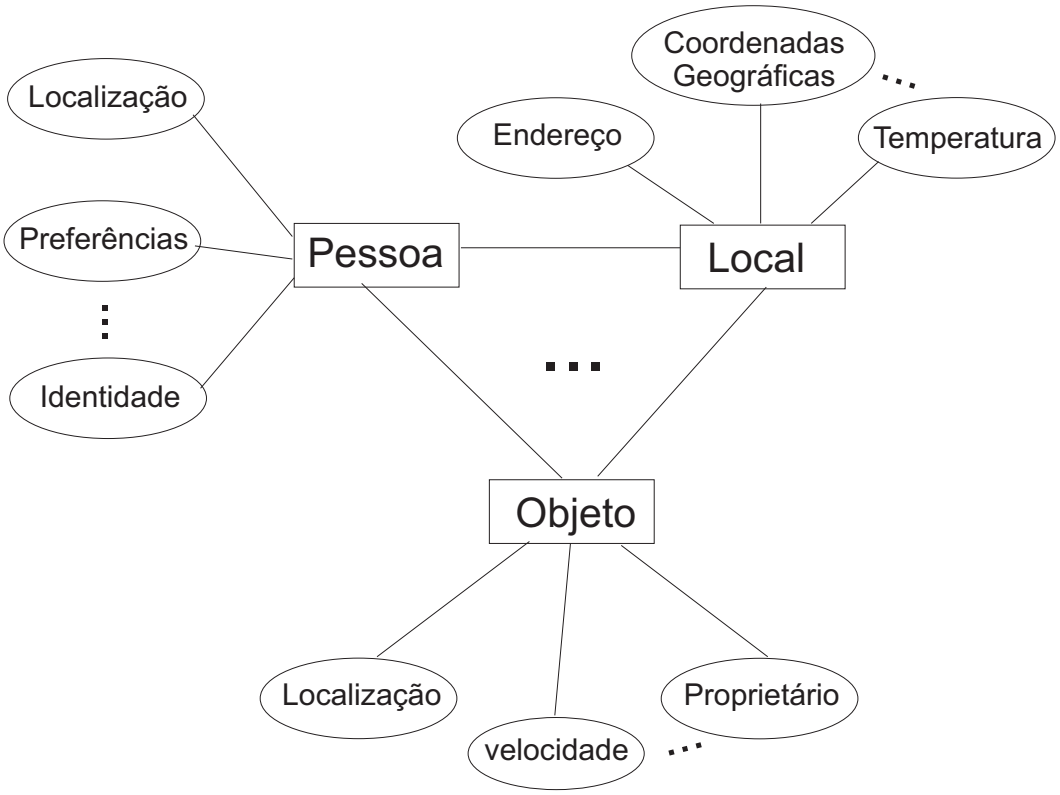

Figure 1.3. A definição de contexto segundo Dey et al.

fined context, por outro lado, é tipicamente "definido" pelo próprio usuário, como seus restaurantes ou produtos preferidos.

O *contexto indireto* é derivado através da interpretação de contexto direto (*context reasoning*). Por exemplo, o estado atual de uma pessoa (tomando banho) pode ser inferido de sua localização (está no banheiro), o estado do chuveiro (ligado) e o estado da porta do banheiro (fechada).

A Figura 1.4 exibe os diferentes tipos ou categorias de contexto discutidas por T. Gu *et al.*

De modo diferente, Schilit *et al.* [10] descreve a existência de três categorias de contexto, a saber:

Contexto Computacional, ou seja, uso de CPU e memória, largura de banda, custos de comunicação, conectividade da rede e outros.

Contexto de Usuário que engloba questões como localização, perfil do usuário, situação social atual.

Contexto Físico, isto é, luminosidade, níveis de ruído, condições de tráfego e temperatura, dentre outros.

A Figura 1.5 descreve essas três categorias.

Guanling Chen e David Kotz [11] ainda propõem uma outra categoria, o *contexto*

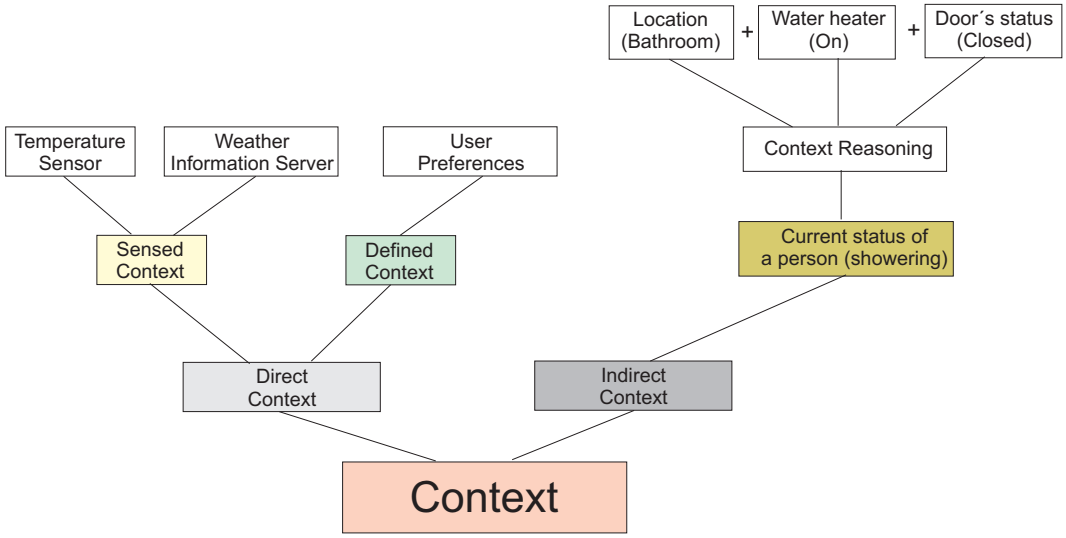

Figure 1.4. Os diferentes tipos ou categorias de contexto descritas por T. Gu *et al.*

temporal, tal como a hora do dia, semana, mês e, até mesmo, a estação do ano.

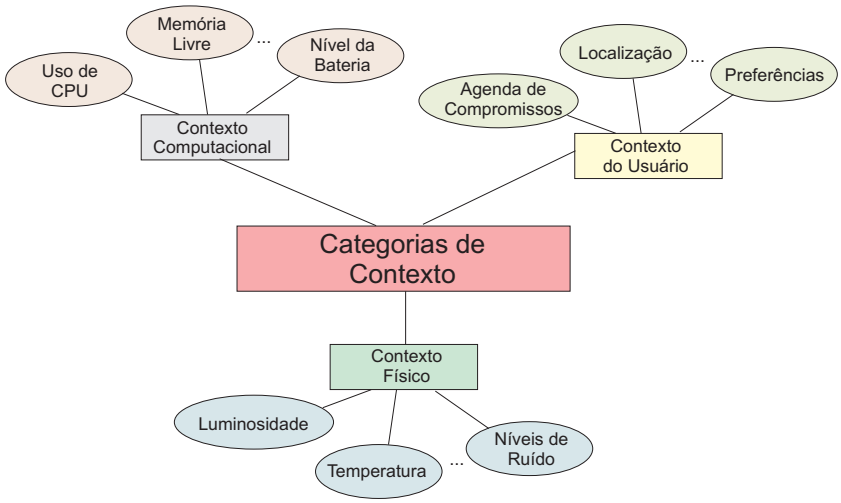

Figure 1.5. As três categorias de contexto descritas por Schilit *et al.*

1.3.3. Sistemas Sensíveis ao Contexto

Em geral, os organismos vivos são dotados de mecanismos de sensibilidade que permitem coletar informações sobre o ambiente que os envolve. Insetos, plantas e, especialmente, o ser humano dependem dos orgãos de sentidos (tato, olfato, paladar, visão e audição) para perceber o meio em que vivem e, em seguida, reagir da forma mais adequada a esse ambiente, que pode ser hostil, agradável, perigoso, etc. Esta capacidade de perceber o mundo e reagir a suas mudanças é vital para a sobrevivência dos seres vivos e, sem dúvida, pode ser vital para a sobrevivência das aplicações computacionais do Século XXI.

Por isso, é cada vez maior a importância dada à construção de sistemas computa-

cionais verdadeiramente sensíveis ao contexto. Para Seng Loke [3], um Sistema Pervasivo Sensível ao Contexto deve ter três funcionalidades básicas: *Sensibilidade*, *Processamento* e *Execução de ações*. De acordo com ele, os sistemas podem até variar o grau de sofisticação em cada uma dessas funcionalidades, mas todas elas devem sempre existir. A Figura 1.6 ilustra esta divisão.

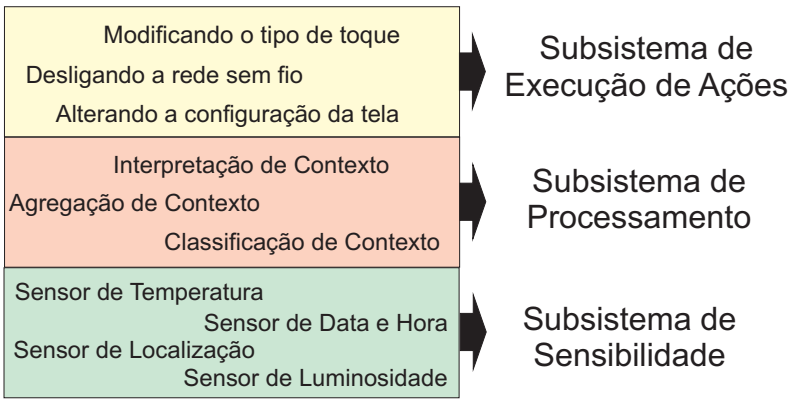

Figure 1.6. A estrutura interna básica de um sistema sensível ao contexto

1.3.3.1. Sensibilidade (*Sensing*)

De fato, os sensores são os mecanismos responsáveis por coletar ou adquirir dados ou informações sobre o mundo físico ou algum aspecto deste. Existem sensores biológicos e não biológicos, os quais podem ser combinados para tornar a compreensão do mundo físico ou de parte dele o mais coerente possível com a realidade.

Além disso, é possível e, até, desejável em muitos casos o uso de múltiplos sensores a fim de fornecer uma maior quantidade de informações relevantes às aplicações sensíveis ao contexto, permitindo processamento e conclusões mais complexas e mais reais. Seng Loke [3] considera os sensores como uma ponte entre o mundo físico e o mundo virtual dos programas de computador. Para ele, esta é a grande diferença entre um software convencional e um sistema sensível ao contexto, visto que em um software convencional a entrada de dados normalmente deve vir manualmente através dos usuários. Porém, em um sistema sensível ao contexto, os sensores são os maiores responsáveis pela entrada de informações, automatizando ao extremo a execução de aplicações.

Deve-se observar também que tipo de informação pode ser coletada por um sensor. Nos últimos anos, com o desenvolvimento da tecnologia da informação, microeletrônica e outras, várias espécies de sensores estão sendo desenvolvidas, como sensores de movimento, localização, temperatura, luminosidade, etc. Além desses, uma grande quantidade de dispositivos podem ser considerados sensores, como microfones, medidores de pressão e relógios internos de computadores (*computer clock*) [3].

De forma simplificada, pode-se dividir os sensores em dois grandes grupos: os *sensores físicos* e os *sensores lógicos*. Os sensores físicos são os dispositivos físicos (hardware) capazes de coletar dados de alguma variável física do ambiente, como locali-

zação, luminosidade, umidade, pressão atmosférica, temperatura, dentre outras. Por outro lado, os sensores lógicos, são, na verdade, aplicações computacionais (software) capazes de coletar alguma informação que, em geral, não pode ser coletada por dispositivos físicos, como a agenda de compromissos de um usuário, suas preferências, suas páginas Web mais visitadas, dentre outras.

Devido à grande diversidade de informações de contexto, há uma enorme quantidade de sensores existentes e um dos desafios ou preocupações dos desenvolvedores de aplicações sensíveis ao contexto em ambientes ubíquos é a localização desses sensores. Os sensores podem ser instalados no próprio ambiente físico (como sensores de luminosidade, temperatura, nível de ruído, etc.), em automóveis, nos dispositivos móveis, como PDAs e *smartphones*, que acompanham os usuários e, até mesmo, serem instalados em roupas, na pele ou dentro dos próprios usuários, como sensores médicos cardíacos, pílulas inteligentes, e outras.

Finalmente, é importante destacar que, quase sempre, é útil esconder como o tipo de informação de contexto foi adquirida, pois isso torna as aplicações que usam tais informações menos sensíveis às mudanças na forma como essas informações de contexto são coletadas.

1.3.3.2. Processamento (*Thinking*)

Após os dados contextuais terem sido coletados pelos sensores, eles podem ser imediatamente utilizados pelas aplicações ou serem processados para que novas formas de conhecimento possam ser extraídas desses dados brutos e serem utilizadas de forma mais eficiente pelas aplicações. De fato, os dados e informações capturados pelos sensores podem não ser suficientes para auxiliar as aplicações e os usuários em suas atividades. Muitas vezes, é preciso agregar informações de contexto mais simples ou até mesmo inferir informações de contexto mais complexas a partir de dados mais simples para que estas possam, de fato, serem úteis às aplicações.

De modo geral, há três tipos principais de processamento de dados coletados pelos sensores: classificação, agregação e interpretação.

A *interpretação de contexto* tem a função de inferir novas informações contextuais a partir de outras já existentes. Isto é, obtém informações contextuais de mais alto nível a partir de dados de contexto de mais baixo nível. Por exemplo, um interpretador de contexto poderia, a partir das coordenadas geográficas fornecidas por um receptor GPS, inferir se um usuário específico se encontra ou não em sua residência ou em seu escritório num determinado momento.

Já a *agregação de Contexto* tem a responsabilidade de reunir informações contextuais inter-relacionadas, permitindo que estas informações possam ser manipuladas como uma única unidade. Como exemplo, pode-se citar um conjunto de informações de contexto relativas a uma determinada sala. Nela, há vários sensores coletando dados sobre temperatura, luminosidade, ruído e umidade. Por serem informações fornecidas por diferentes fontes contextuais, estes dados são tratados internamente como informações disjuntas. Um *agregador de contexto* poderia reunir todos esses dados em uma única

unidade de contexto relacionada à sala de onde essas informações estão sendo coletadas, facilitando assim o trabalho de interpretação e utilização desses dados.

A *classificação de contexto* é empregada para organizar os dados contextuais obtidos dos sensores em grupos e subgrupos de modo a facilitar tanto o trabalho dos agregadores quanto dos interpretadores de contexto. Pode-se, por exemplo, classificar as informações de contexto através das entidades (pessoas, objetos, locais, etc.) às quais tais informações então associadas, ou ainda classificar todas as informações de contexto relativas a um determinado usuário (localização, pessoas colocalizadas, preferências, etc.). Não se deve confundir a classificação de contexto com a agregação de contexto. Classificar é realizar um processamento prévio nas informações, de modo a tornar mais fácil e rápido a identificação dessas informações, através de vários aspectos (por entidades, por precisão, etc.). Entretanto, agregar é reunir várias informações relacionadas a uma única entidade, por exemplo, para que posteriormente se possa manipulá-las como uma única informação composta.

Deve-se ressaltar, além disso, que muitas vezes, é necessário persistir os dados de contexto, de modo a oferecer dados históricos que possam ser úteis às aplicações posteriormente. Por exemplo, pode ser importante para uma determinada aplicação, ou mesmo para um interpretador de contexto, analisar as variações que uma determinada variável contextual sofreu durante um período específico, permitindo, a partir desses dados, fazer previsões sobre o que poderia acontecer ao ambiente em um intervalo de tempo subsequente.

1.3.3.3. Execução de Ações (*Acting*)

Uma vez que as informações de contexto são colhidas ou algumas situações são reconhecidas, ações devem ser executadas [3]. É importante notar que as ações a serem executadas são específicas de cada aplicação. Muitas vezes, essas ações devem ser executadas imediatamente, antes mesmo que as situações que causaram a necessidade da execução dessas ações sejam alteradas. Como exemplo, imagine uma aplicação baseada em localização utilizada para notificar usuários de dispositivos móveis sobre a presença de lojas que ofereçam determinados produtos de interesses para estes usuários. Se enquanto o usuário estiver se movimentando com seu automóvel, a aplicação descobrir alguma loja que ofereça o produto desejado pelo usuário, esta deverá avisá-lo imediatamente, pois, caso contrário, se houver demora nessa notificação, esta informação poderá já não ser tão útil caso o usuário já se encontre em uma outra região da cidade.

1.3.3.4. Desenvolvendo uma arquitetura para sistemas sensíveis ao contexto

Durante os últimos anos, várias propostas de arquiteturas para sistemas sensíveis ao contexto foram desenvolvidas. A Figura 1.7 mostra uma arquitetura abstrata em camadas proposta por Baldauf e Dustdar [12]. Para eles, o *subsistema de sensibilidade* é composto pelos sensores propriamente ditos e por uma camada superior de recuperação de dados brutos, ou seja, ainda sem processamento algum. O *subsistema de processamento* é constituído pelas camadas de pré-processamento (em que os dados brutos são organizados) e o

pelo processamento propriamente dito; além disso, há uma camada superior responsável pelo armazenamento e gerenciamento das informações processadas pela camada inferior. Finalmente, as aplicações sensíveis ao contexto compõem o *subsistema de execução de ações*, onde, de fato, todas as ações ou reações ocasionadas pelas respectivas mudanças no ambiente que envolve as aplicações são executadas.

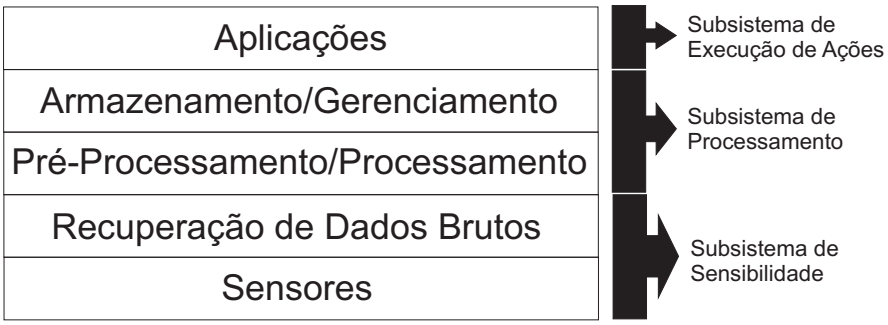

Figure 1.7. Uma arquitetura abstrata em camadas para sistemas sensíveis ao contexto.

1.3.4. Infraestruturas para Provisão de Contexto

A provisão e o processamento de informações de contexto são tarefas difíceis e complexas. Esta afirmação se torna mais clara à medida que verificamos a diversidade de fontes de contexto (sensores) e a heterogeneidade das informações de contexto disponíveis ou necessárias, bem como dos ambientes de execução existentes.

Para solucionar ou, pelo menos amenizar, o problema anterior, há uma grande quantidade de pesquisas cuja finalidade principal é o desenvolvimento de camadas de software e hardware capazes de atuarem como intermediárias entre as aplicações e os sistemas computacionais para obtenção, tratamento e disponibilização das informações de contexto. Essas camadas intermediárias são conhecidas como *infraestruturas para provisão de contexto*.

1.3.4.1. Context Toolkit

O Context Toolkit [13] [14] é, na verdade, uma implementação de um *framework* conceitual ou, como os próprios autores dizem, um kit de ferramentas para dar suporte ao desenvolvimento rápido de aplicações sensíveis ao contexto.

Este *framework* conceitual é composto de elementos que atuam desde a aquisição até o processamento e distribuição de informações de contexto às aplicações. Seus elementos são: *context widgets*, interpretadores (*interpreters*), agregadores (*aggregators*), serviços (*services*) e descobridores (*discoverers*). A seguir, cada elemento será examinado em maiores detalhes.

Context widgets: são componentes de software empregados para fornecerem às aplicações acesso à informação de contexto, ou seja, são mediadores localizados entre a aplicação e seu ambiente operacional. Suas principais funções são:

Fornecer uma separação de interesses, escondendo da aplicação a complexidade dos sensores reais utilizados;

Fornecer informações abstratas de forma a alimentar a aplicação com apenas informações relevantes;

Fornecer blocos de aquisição de contexto reusáveis e personalizáveis que possam ser utilizados por uma variedade de aplicações sem a necessidade de serem alterados.

Interpretadores (Interpreters): são os responsáveis por elevar o nível de abstração de um certo "bloco" de contexto. Ou seja, um interpretador de contexto recolhe informações de uma ou mais fontes de contexto e produz uma nova unidade de informação contextual. Todos os interpretadores têm uma interface comum facilitando, assim, a identificação das capacidades que um determinado interpretador fornece, além de permitir a comunicação com o ambiente externo.

Agregadores (Aggregators): têm a função de reunir múltiplos pedaços de informação de contexto que estão logicamente relacionados e armazená-los em um repositório comum. De acordo com os autores, a necessidade de agregação surge devido principalmente à natureza distribuída da informação de contexto.

Serviços (Services): Nota-se que os três primeiros componentes citados (widgets, interpretadores e agregadores) são responsáveis pela aquisição de contexto e sua respectiva entrega às aplicações interessadas nesses dados. Porém, os serviços "são os componentes no *framework* que executam as ações em favor das aplicações" [14]. Para os autores, estes serviços podem ser síncronos ou assíncronos, dependendo ou não da necessidade de se esperar uma resposta para que outras ações sejam executadas subsequentemente.

Descobridores (Discoverers): destinam-se a manter um registro de quais capacidades existem, em cada momento, no *framework*. Por esta razão, são considerados os componentes finais no *framework* conceitual proposto. Portanto, os *descobridores* devem conhecer quais widgets, interpretadores, agregadores e serviços estão atualmente disponíveis para uso pelas aplicações. A Figura 1.8 exibe o diagrama de objetos para as abstrações descritas anteriormente para o Context Toolkit.

Segundo este modelo proposto, "um número limitado de relacionamentos podem ser estabelecidos entre esses componentes. Widgets podem ser consultados ou usarem notificações para informar a seus clientes sobre mudanças ocorridas. Seus clientes podem ser aplicações, agregadores ou outros widgets. Agregadores, normalmente, atuam como pontes entre widgets e as aplicações. Interpretadores podem ser solicitados em qualquer estágio e por qualquer widget, agregador ou aplicação. Serviços são acionados primeiramente pelas aplicações (embora outros componentes possam também usá-los). E, finalmente, descobridores se comunicam com todos os componentes, adquirindo informação de widgets, interpretadores e agregadores e fornecendo estas informações às aplicações via notificações ou consultas." [14].

Como já foi citado anteriormente, o Context Toolkit é uma implementação deste *framework* conceitual discutido nos parágrafos anteriores. Esta implementação foi desenvolvida em Java, embora, de acordo com seus autores, tenham sido usados mecanismos

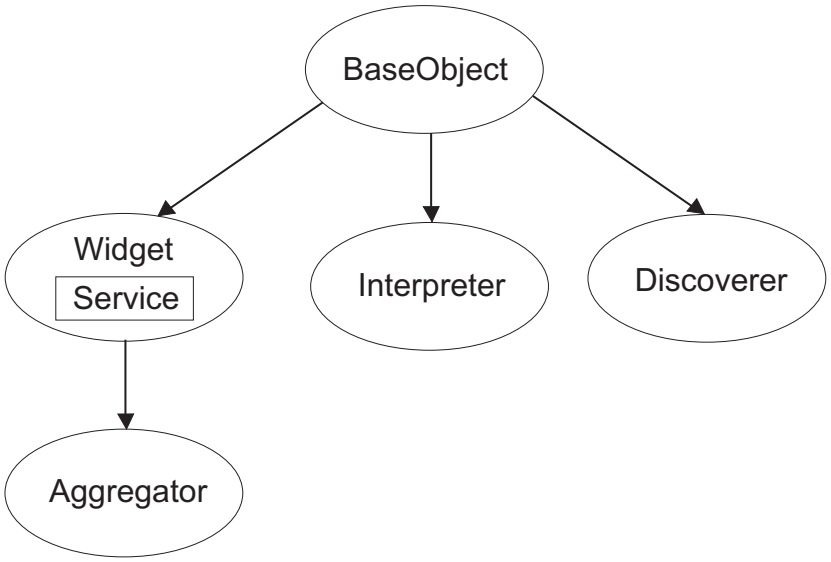

Figure 1.8. Diagrama de objetos para as abstrações do Context Toolkit.

independentes de linguagem de programação.

É importante notar que, nessa implementação, cada componente (widget, interpretadores, agregadores e descobridores) é implementado com um único processo. No Context Toolkit, componentes diferentes podem estar distribuídos em processadores diferentes. Por esta razão, este *framework* é construído no topo de uma simples infra-estrutura distribuída utilizando comunicações ponto-a-ponto.

Comunicação Distribuída

Para a comunicação distribuída, foi empregado um mecanismo de comunicação de objetos simples baseada em HTTP (HyperText Transfer Protocol) e XML (eXtensible Markup Language) para codificar as mensagens. A habilidade da comunicação distribuída, nesse caso, é encapsulada em um componente denominado BaseObject. Por isso, tanto os widgets quanto os interpretadores são implementados como subclasses do BaseObject; porém, os agregadores são subclasses dos widgets. Esta hierarquia permite que todos esses componentes possuam habilidades de comunicação embutidas.

Processo de Inscrição para Recebimento de Contexto

Para que uma aplicação possa se inscrever em um context widget, ela deverá primeiramente adquirir um manipulador (handle) para o widget correspondente. Isso é feito contactando o descobridor e especificando qual tipo de informação de contexto é necessária. De posse do manipulador do widget desejado, pode-se usar sua instância do BaseObject para contactar o widget. Ao receber a comunicação, o BaseObject do widget decodifica a mensagem e a entrega ao widget propriamente dito. Este adiciona a aplicação em sua lista de aplicações interessadas em seu contexto juntamente com todos os dados para uma posterior comunicação (handle da aplicação, por exemplo). Desse modo, quando um widget recebe nova informação de contexto de seu sensor, ele recorre à sua lista de aplicações cadastradas para enviar o contexto apropriado a cada uma delas. O

contexto é enviado pelo widget usando sua instância do BaseObject que se comunicará diretamente com o BaseObject da aplicação. Então, o BaseObject da aplicação decodificará a mensagem enviada e chamará, em seguida, o método apropriado da aplicação.

Manipulando Eventos

Para se entender como os eventos são manipulados no Context Toolkit deve-se observar que quando alguma informação de contexto é modificada, esta modificação é detectada pelo context widget responsável pela gerência deste pedaço de informação. Em seguida, o widget responsável gera uma marca de tempo indicando o momento da alteração e então notifica todos os seus clientes inscritos em sua lista de interessados naquela informação de contexto (widgets, agregadores e aplicações).

Frequentemente, a nova informação é modificada pelos interpretadores encontrados pelo caminho entre o widget e o cliente. Pelo fato de que o widget e seus clientes não necessariamente estarem executando na mesma máquina, os eventos são enviados através da infra-estrutura distribuída como mensagens XML sobre o protocolo HTTP.

Mecanismo de Descoberta

É importante observar que o mecanismo de descoberta usado no Context Toolkit é centralizado. Ou seja, é empregado apenas um único descobridor e não uma federação de descobridores discutida no *framework* conceitual. O processo todo funciona da seguinte maneira:

Quando iniciados, todos os widgets, agregadores e interpretadores se registram com o descobridor em um endereço de rede e uma porta conhecidos por todos os componentes. Logo após, o descobridor verifica se cada componente registrado está funcionando adequadamente. Caso durante este teste algum componente falhe, o componente é removido da lista de registros e outros componentes que registraram interesse neste componente específico são notificados. Além disso, sempre que algum componente é desativado manualmente, eles próprios notificam o descobridor indicando que já não podem mais serem utilizados.

Serviços de Contexto

No Context Toolkit, os serviços são incorporados nos widgets. Portanto, além de um widget ser responsável pela coleta de dados de um sensor, ele também tem a habilidade de executar ações no ambiente em que está inserido. Por exemplo, um Widget Temperatura pode não apenas indicar a temperatura corrente emitida por um sensor sob sua responsabilidade como também alterar a quantidade de vapor da caldeira em que está inserido seu sensor.

1.3.4.2. Uma Aplicação-Exemplo

A Figura 1.9 mostra o diagrama arquitetural para a aplicação Guia Turístico Móvel (Mobile Tour Guide). Esta é uma das aplicações sensíveis ao contexto mais comuns e permite a um visitante percorrer ambientes internos e externos desconhecidos. À medida em que o visitante se move, a aplicação exibe na tela do dispositivo móvel a sua localização corrente em um mapa e exibe ainda informações sobre os locais interessantes que estão próximos

a ele.

Segundo os autores [14], para esta aplicação deve-se assumir que ela deverá ser executada em um ambiente interno (indoor space) e que este ambiente possui um sistema de localização que relata atualizações de localização para um servidor centralizado. O *Location Widget* exibido no diagrama arquitetural é construído sobre um servidor de localização centralizado. Cada *Agregador de Sala* ou *Room Aggregator* (um para cada sala) se registra no *Location Widget* para ser notificado quando um usuário entrar ou sair de uma sala correspondente. Os agregadores de sala também se registram nos *Exhibit Widgets* (Widgets de Exibição de Imagens dos ambientes) em suas respectivas salas, assim ficam sabendo quais imagens a serem exibidas estão disponíveis em um dado instante. Cada *Exhibit Widget* contém informações sobre um ícone (representação de um objeto real) na rota. A aplicação se registra no *Location Widget* para atualizações de localização para seus usuários. Quando recebe uma atualização, ela atualiza seu mapa e consulta o *Agregador de Sala* correspondente à localização corrente do usuário para verificar a lista de ícones de exibição disponíveis, se existirem. Em seguida, exibe o ícone correspondente ao usuário e também se registra junto ao *Agregador de Sala* para ser notificado sobre quaisquer mudanças na lista de ícones de exibição.

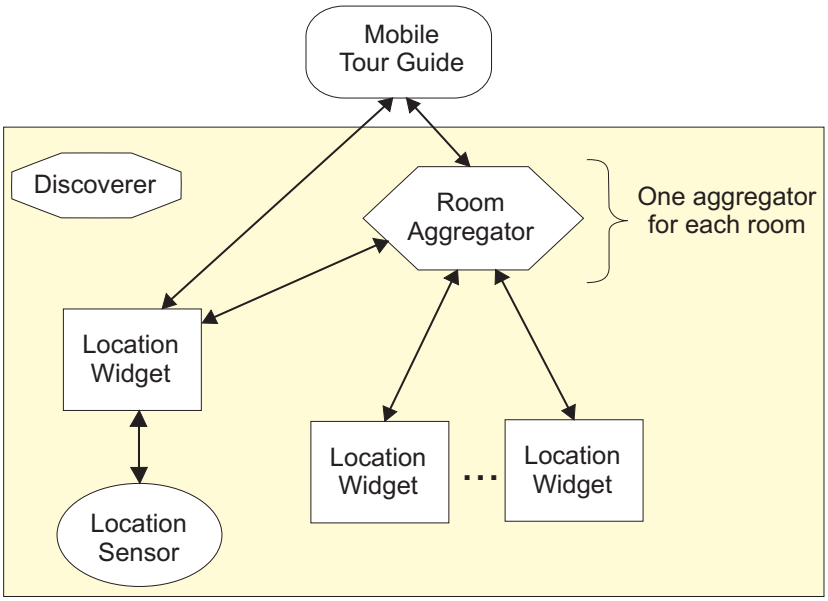

Figure 1.9. Diagrama arquitetural para a aplicação Guia Turístico Móvel.

1.3.5. ConBus

O *ConBus* [15] é uma arquitetura para provisão de contexto em dispositivos móveis. Conforme ilustrado na Figura 1.10, ela é constituída por três camadas: *Camada de Sensores*, *Framework ConBus* e *Camada de Aplicação*.

O *framework ConBus*, que implementa a parte intermediária da arquitetura, executa, de forma co-localizada com as aplicações, no dispositivo móvel. Dessa forma, mais de uma aplicação, executando no mesmo dispositivo móvel, poderá utilizar a mesma instância do *framework ConBus* para obter dados de contexto desejados, desde que sejam ins-

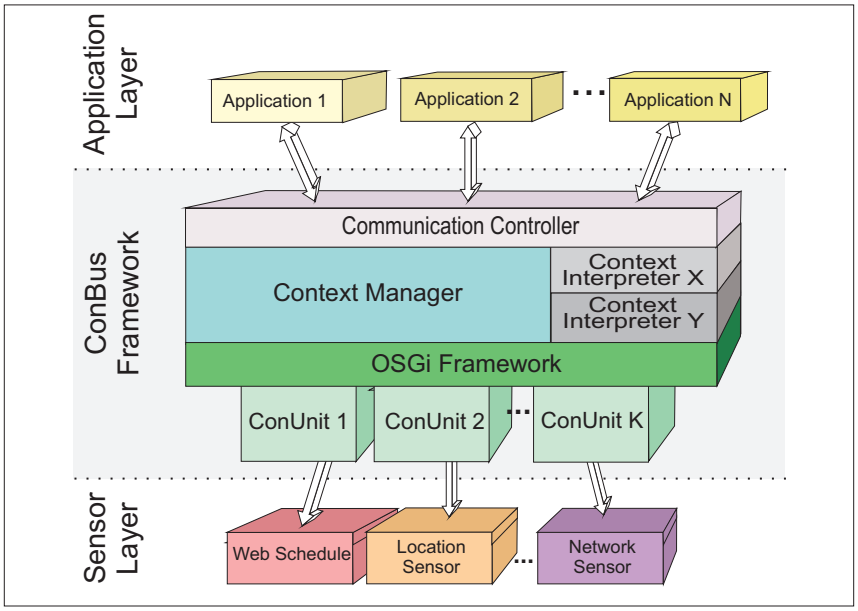

Figure 1.10. A arquitetura ConBus - Modelo Geral.

talados os módulos de sensoriamento adequados a cada uma delas. Este *framework* oferece às aplicações sensíveis ao contexto, através de uma interface de acesso padronizada (descrita na Seção 1.3.5.3), acesso a todas as informações contextuais providas pelos sensores acoplados ao *Framework*. Deve-se observar que, visando minimizar o tempo e o custo de se desenvolver novas aplicações sensíveis ao contexto, um desenvolvedor de uma aplicação móvel poderá, ainda, acoplar ao *ConBus* outros módulos de sensoriamento existentes sempre que houver necessidade, desde que esses módulos de sensoriamento tenham sido desenvolvidos de acordo com esta arquitetura.

1.3.5.1. Camada de Sensores

A Camada de Sensores (*Sensor Layer*) ilustrada na Figura 1.10 é constituída por todas as fontes de contexto (sensores) que fornecem informações ao *framework ConBus*.

Em princípio, além dos sensores físicos, como aqueles que coletam dados sobre temperatura, localização, luminosidade, nível de energia, etc; outros tipos de sensores, denominados sensores lógicos, podem ser incorporados à arquitetura ConBus. Estes sensores podem coletar dados de contexto, como compromissos de agendas eletrônicas, preferências de usuários, páginas Web visitadas, dentre outros.

1.3.5.2. Framework ConBus

O *Framework ConBus* é a camada intermediária que gerencia todas as informações contextuais coletadas pelas fontes de contexto da *Camada de Sensores* e que, posteriormente, são fornecidas a todas as aplicações sensíveis ao contexto que fazem uso das mesmas. Esta camada é composta pelos seguintes componentes: o Gerenciador de Contexto (*Con-*

text Manager), responsável pelo controle de todos os recursos oferecidos pelo *framework*, ou seja, o controle de todas as informações coletadas pelos sensores acoplados ao *ConBus*, bem como a responsabilidade de disponibilizá-las adequadamente às aplicações sensíveis ao contexto. Além do Gerenciador de Contexto, há também os *ConUnits* (Unidades de Contexto) que interligam, de forma padronizada, as fontes de contexto ao *ConBus*; há também o Controlador de Comunicações (*Communication Controller*) cuja função principal é permitir a comunicação entre o *framework ConBus* e as aplicações sensíveis ao contexto.

Outros componentes importantes são os Interpretadores de Contexto (*Context Interpreters*), que realizam tarefas importantes para a arquitetura, inferindo novas informações de contexto a partir de dados de contexto de mais baixo nível, como, por exemplo, a indicação de que um usuário se encontra ou não em uma reunião, inferida a partir de sua localização, data e hora correntes e as informações de sua agenda de compromissos.

Além dos componentes citados anteriormente, há ainda o *framework OSGi* [16] cuja função é permitir o acoplamento de novos *ConUnits* e *Interpretadores de Contexto* ao *Context Manager*. O *OSGi* permite ainda que *ConUnits* ou *Interpretadores de Contexto* sejam instalados, desistalados, iniciados, desligados, e até, atualizados dinamicamente sem a necessidade de reiniciar a execucão de todo o *framework ConBus*. Isto permite que o próprio *ConBus* seja sensível ao contexto e, sempre que necessário, possa instalar ou desistalar, iniciar ou desligar *ConUnits* ou Interpretadores de Contexto, visando poupar recursos computacionais, como, baixo nível de energia, alto uso de CPU e memória, etc.). Além disso, pode-se realizar, remotamente e de forma automatizada, atualizações em *ConUnits* e/ou *Interpretadores de Contexto*.

1.3.5.3. Camada de Aplicação

A Camada de Aplicação (*Application Layer*) é constituída por todas as aplicações sensíveis ao contexto que utilizam as informações contextuais fornecidas pelo *framework ConBus*. Para usufruir das informações de contexto providas pelo *ConBus*, uma aplicação necessita apenas conhecer a URL (neste caso, http://localhost:8585) da instância do *framework ConBus* que está sendo executada no dispositivo móvel correspondente e, em seguida, invocar os métodos desejados com os devidos parâmetros oferecidos pela interface de acesso ao contexto, usando, para isso, interfaces síncronas e assíncronas providas pela API das Aplicações (*Application API*). Esta API define um *Web Service* local baseado em REST [17], através do qual as aplicações podem enviar requisições para a instância do *ConBus* que se encontra em execução.

A comunicação síncrona ocorre da seguinte forma: como dito anteriormente, o *ConBus* exporta um *Web Service* local (i.e., associado ao endereço *localhost*) em uma porta pré-definida (a padrão é 8585) para o qual as aplicações podem enviar requisições utilizando o comando *GET* e informar, como parâmetros, o nome da variável de contexto na qual está interessada (LOCATION, TEMPERATURE, ENERGYLEVEL, etc.) e a entidade a qual essa variável está relacionada (ex. CARLOS14352, SALA113, NokiaE51-1213, etc.). Como resposta, recebem a informação de contexto correspondente, representada em um formato XML predefinido, de acordo com o modelo de contexto discutido na

Subseção 1.3.5.4.

A comunicação assíncrona acontece de forma semelhante, porém, deve-se enviar requisições utilizando-se o comando *POST*, seguido pela palavra *Assinc* e informando, em seguida, o *nome da variável de contexto* desejada juntamente com uma *expressão* que contém uma condição ou restrição e, ainda, o *nome da entidade* a qual essa variável está relacionada (por exemplo, ENERGYLEVEL $\leq$ 25%, iPhone2132) para que a instância do *ConBus* em execução envie as informações contextuais das quais a aplicação está interessada.

Nota-se que, à semelhança com o Context Toolkit, apresentado na Subseção 1.3.4.1, apesar da instância do *ConBus* e as aplicações estarem executando localmente no mesmo dispositivo móvel, a comunicação entre eles ocorrerá via interface de rede, neste caso, um *Web Service* local; segundo os autores, isto foi proposto para tornar o uso do *ConBus* independente da linguagem na qual a aplicação foi desenvolvida.

1.3.5.4. Modelagem de Contexto

A arquitetura *ConBus* usa um modelo de contexto baseado na modelagem contextual discutida em [18]. Ele é constituído por cinco elementos principais, a saber: *Context*, *ContextInformation*, *ContextType*, *ContextData* e *Entity*. A Figura 1.11 exibe o diagrama de classes UML para o modelo proposto pelo *ConBus*.

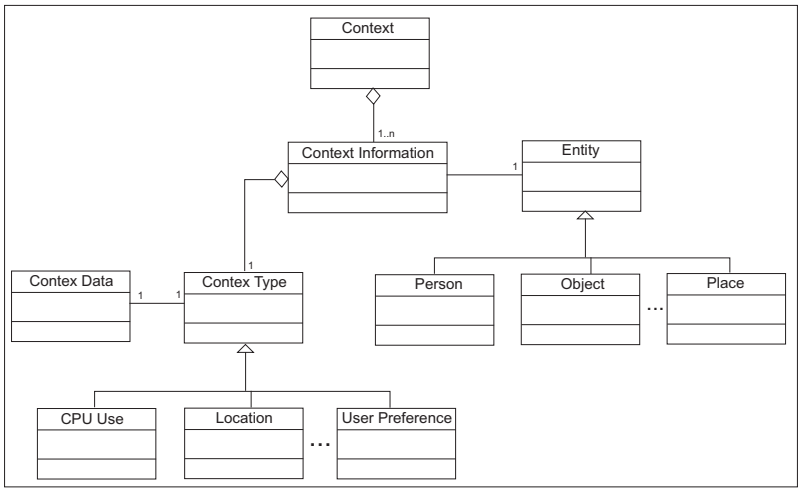

Figure 1.11. Modelagem de Contexto empregada pela arquitetura ConBus.

Context representa toda a informação de contexto gerenciada pelo *ConBus* em um dado momento. É constituído por uma ou mais informações de contexto (*ContextInformation*). Cada informação de contexto armazena internamente um tipo de contexto específico (*ContextType*), por exemplo, localização (*Location*), temperatura (*Temperature*), etc., através de suas meta-informações e dados de contexto (*ContextData*) propriamente ditos. Para o *ConBus*, cada informação de contexto (*ContextInformation*) deve estar relacionada a uma única entidade (*Entity*). Uma entidade pode ser, por exemplo, uma pessoa (*person*), um objeto (*object*), um local (*place*), dentre outros.

O elemento *ContextData* representa os dados de contexto propriamente ditos provenientes de algum sensor (lógico ou físico). Ou seja, se um sensor coleta informações sobre localização, por exemplo, os valores das coordenadas geográficas, juntamente com suas unidades e outros dados intimamente relacionados a essa informação de contexto deverão estar representados nestes elementos. A Figura 1.12 ilustra o uso do elemento *ContextData*.

Context Data			
Value:	16.2989443,	27.3849485,	2000
TypeOfData:	latitude,	longitude,	altitude
Unit:	degree,	degree,	meter
DataType:	double,	double,	float

Figure 1.12. Exemplo de uso do elemento ContextData.

1.3.6. MoCA - Mobile Collaboration Architecture

A arquitetura MoCA [19, 20] auxilia o desenvolvimento de aplicações colaborativas móveis através de serviços de coleta, agregação e difusão de informações de contexto computacional e de localização. A MoCA consiste de serviços de provisão contexto que disponibilizam para as aplicações o contexto da rede e do dispositivo, de APIs de comunicação e um framework (PrFw) para auxiliar a integração das aplicações com tais serviços através de uma comunicação síncrona ou assíncrona (baseada em eventos).

Uma visão geral dos serviços da MOCA é ilustrada na Figura 1.13.

No projeto da MoCA, é definida uma separação de tarefas em serviços distintos, onde cada serviço implementa uma funcionalidade específica para a coleta ou divulgação do contexto. Por um lado, essa decisão de projeto favoreceu a flexibilidade no uso dos serviços, pois o desenvolvedor pode utilizar somente aqueles de que realmente necessita. Além disso, essa abordagem permite que novos serviços possam ser incorporados à arquitetura de forma modular, tornando os serviços do núcleo da MoCA mais escaláveis por não sobrecarregá-los com a implementação das funcionalidades envolvidas na provisão dos mais diversos tipos de informação de contexto (e.g, contexto computacional, contexto pessoal). Por outro lado, essa decisão acarreta um forte acoplamento entre os serviços de provisão de contexto que precisam interagir uns com os outros para desempenhar suas funções. Por exemplo, o LIS usa no cálculo da inferência da localização as informações de RSSI providas pelo CIS, que, por sua vez, foram coletadas pelos Monitoresexecutando nos dispositivos móveis dos usuários. Esse acoplamento configura uma dependência de execução entre os serviços. Ou seja, para utilizar o LIS, o CIS deve estar executando. Para auxiliar a implantação e uso desses serviços, é descrito em [20] um documento (*MoCA Overview*) que discute as questões de dependências de execução e implementação entre as APIs e serviços, e os passos necessários para desenvolver uma aplicação baseada na MoCA.

Os serviços de provisão de contexto da MoCA são brevemente descritos nas subseções seguintes. Maiores detalhes sobre como esses interagem com os demais serviços da arquitetura podem ser encontrados em [20, 19]. O CIS, o MONITOR e as APIs de comunicação síncrona e assíncrona (baseadas em eventos) constituem parte da contribuição

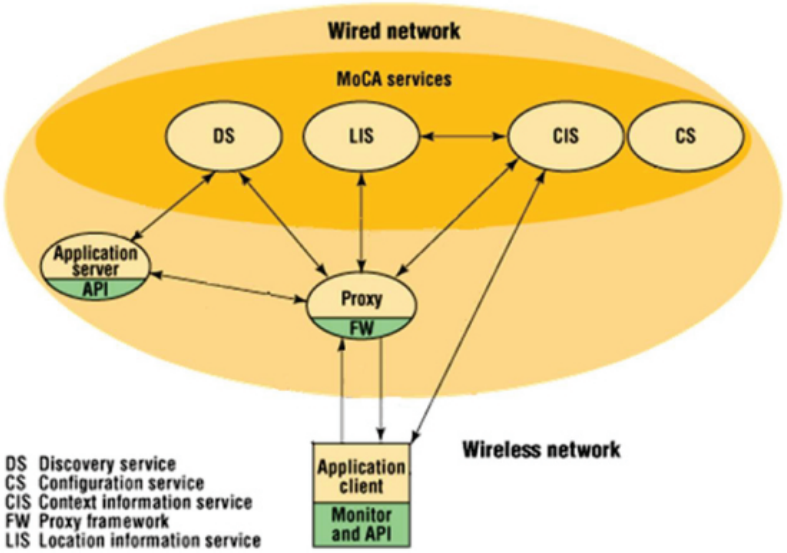

Figure 1.13. Arquitetura MoCA

deste trabalho. O serviço de localização (LIS) que foi o principal elemento usado no estudo de caso para o controle de privacidade, no entanto, foi resultado de uma dissertação de mestrado [21].

1.3.6.1. Monitor

O Monitor coleta informações do dispositivo móvel e da rede sem fio e as envia periodicamente para o CIS na rede fixa. Tais informações (na forma de uma lista de variáveis) podem ser classificadas em duas categorias, como a seguir.

Informações coletadas da rede sem fio:

- intensidade do sinal e endereço MAC de todos os *Access Points* (APs) IEEE 802.11 que estão no raio de cobertura do dispositivo;
- endereço MAC e intensidade do sinal do AP corrente.

Informações coletadas sobre a configuração e os recursos disponíveis do dispositivo móvel:

- taxa de uso da CPU (em %);
- memória disponível (em Kbytes);
- endereço MAC e IP/Máscara;
- nível de energia disponível (em %).

Além dos envios periódicos das informações supracitadas, o MONITOR também as enviará quando for detectada uma mudança no endereço IP ou *AP* corrente, já

que isso caracteriza um *roaming* (*handover*) e, portanto, representa uma mudança do estado/contexto corrente do dispositivo que possivelmente é de interesse para as aplicações. As informações coletadas pelo MONITOR podem ser utilizadas para diferentes propósitos, como, por exemplo, implementar uma adaptação dinâmica das aplicações em função do nível de energia ou da memória disponível.

De uma forma geral, o serviço de sensoriamento é um componente chave para o desenvolvimento de qualquer arquitetura de provisão de contexto. No entanto, a implementação deste serviço geralmente demanda um grande esforço de programação por causa da *complexidade* em obter as informações acerca da utilização dos recursos de cada plataforma de *hardware* (e.g., iPAQs, Palms, Laptops), i.e., como obter a taxa de uso da CPU, bateria, ou, obter informações sobre cada possível interface de rede disponível. Como exemplo da complexidade envolvida no desenvolvimento deste serviço, vale destacar que, às vezes, a implementação da coleta de um contexto específico (e.g., uso da CPU ou intensidade de sinal dos APs 802.11) para dispositivos de uma mesma plataforma e de um mesmo fabricante, como, por exemplo, iPAQs da HP, pode variar de acordo com a série do produto ou versão do sistema operacional.

1.3.6.2. Serviço de informação de contexto

O *Context Information Service* (CIS) é um serviço que recebe, processa e divulga para as aplicações interessadas os dados de contexto recebidos de diversos monitores. Através das APIs de comunicação, *Communication Protocol* e *Event-based Communication Interface* (ECI), a aplicação pode interagir com o CIS de duas formas: por meio de uma comunicação síncrona (requisição/resposta) para obter o valor de uma dada variável de contexto ou por meio de uma comunicação assíncrona baseada em eventos para ser notificada sobre uma eventual mudança de valor de alguma variável de contexto. Através das notificações, as aplicações podem manter-se a par das mudanças do estado corrente do dispositivo móvel (e.g., memória livre, nível de energia) ou da rede sem fio (e.g., variação do RSSI, AP corrente).

Para ser notificada, a aplicação deve se registrar no CIS (através de uma *subscription*) passando como parâmetro um tópico ou assunto (Subject) e, opcionalmente, uma expressão (baseada no padrão SQL92) para filtrar os eventos de seu interesse. Através da API *ECI*, o CIS recebe e registra a *subscription* e, eventualmente, publica um evento que satisfaz as propriedades da assinatura de interesse de uma aplicação. Para o CIS, o tópico (i.e., o Subject) é o endereço MAC do dispositivo e as propriedades especificam uma expressão sobre o estado das variáveis de contexto deste dispositivo (e.g., *Roaming, FreeMemory*) nas quais a aplicação está interessada. Uma *subscription*, por exemplo, poderia ser *Subject = {"02:DA:20:3D:A1:2B"}, Properties = {"roaming = True" OR "FreeMem < 15%" OR "CPU > 90%" OR (("OnLine = False") AND ("DeltaT > 15s"))}*. A aplicação recebe então uma notificação sempre que a condição informada na expressão for satisfeita. Essa notificação descreve o valor corrente das *tags/variáveis* de contexto do dispositivo.

A aplicação também pode ser notificada quando a condição especificada através da expressão voltar a ser invalidada, i.e., a ter o valor "falso" (evento conhecido também

como *negação da expressão*). Essa funcionalidade é essencial para as aplicações que implementam algum tipo de adaptação baseada em contexto. Por exemplo, após ser notificada que um determinado estado de alguma *tag* de contexto do dispositivo está abaixo do limiar aceitável (e.g., nível de energia abaixo de 10%), a aplicação pode desempenhar alguma função, que possivelmente tem alto custo computacional, para lidar com a situação corrente, como, por exemplo, ativar uma política de *cache*, compactar ou remover parte dos dados enviados para o cliente executando no dispositivo em questão. Por isto, a aplicação também deve ser notificada quando o referido estado do nível de energia voltar a ficar acima do limiar aceitável. Essa funcionalidade permite que a aplicação pare de executar a adaptação. As possíveis *tags* de contexto que podem ser utilizadas nas expressões de consulta ao CIS são descritas na Tabela 1.1.

Tag	**Valor**	**Descrição**
CPU	int	Uso da CPU (entre 0 e 100%)
EnergyLevel	int	Nível de energia disponível (entre 0 e 100%)
AdvertisementPeriodicity	int	Periodicidade em que o monitor envia suas notificações
APMacAddress	string	Endereço Físico (MAC) do Ponto de Acesso
RSSI	long	Intensidade do sinal para com o AP corrente (em .dBm)
FreeMemory	long	Total de memória disponível (em Kb)
DeltaT	long	Por quanto tempo o dispositivo está off-line (em milisegundos)
OnLine	boolean	Verdadeiro se o dispositivo estiver on-line (i.e com conectividade de rede)
IPChange	boolean	Verdadeiro se o dispositivo mudar de endereço IP
Roaming	boolean	Verdadeiro se o dispositivo implementar uma operação de roaming

Table 1.1. Atuais tags de contexto do CIS

Em linhas gerais, o CIS recebe as informações enviadas pelos Monitores, faz uma análise sintática no conteúdo da mensagem e extrai os dados que descrevem o contexto do dispositivo e da rede. Em seguida, ele publica tais dados, usando a API do serviço de eventos, para as aplicações que tenham se registrado como interessadas em alguma mudança de estado do dispositivo ou da rede. A implementação do CIS, da API ECI e *Communication Protocol* estão disponíveis para *download* em [22, 23, 24].

1.3.6.3. Serviço de inferência de localização

O *Location Inference Service* (LIS) infere e disponibiliza a localização simbólica dos dispositivos móveis em áreas cobertas por *Access Points (APs)* de redes IEEE 802.11. Para isso, ele utiliza a intensidade de sinais RSSI coletados e divulgados pelos monitores e CIS, respectivamente.

Conforme descrito em [21], a inferência é feita a partir da comparação da similaridade entre o padrão de sinal de RF atual (representado por um vetor, onde cada elemento deste vetor corresponde ao sinal de um AP "audível" ao dispositivo) e o padrão (vetor RSSI) medido anteriormente em pontos pré-definidos em uma região com cobertura de rede IEEE 802.11. Portanto, em uma primeira etapa (fase de calibração) as amostras de

vetores RSSI são coletadas em *pontos de referência* bem definidos na(s) área(s) de interesse, e armazenadas em um banco de dados do LIS. Considerando que os sinais de rádio são suscetíveis à intensa variação e interferência, a precisão da inferência da localização depende fortemente do número de APs, do número de pontos de referência escolhidos e do número de amostras de vetores RSSI coletados.

O LIS atende, principalmente, aplicações que necessitam conhecer a posição de um dispositivo em termos de regiões simbólicas (ao invés de coordenadas), onde tais regiões são áreas geográficas não menores do que 1 a $4m^2$. Devido a flutuação intrínseca no sinal de rádio, a inferência da localização baseada em RSSI não é capaz de oferecer resultados com maior precisão. Entretanto, acredita-se que a precisão da localização obtida pelo LIS é suficiente para uma grande classe de aplicações sensíveis à localização.

Ao invés de coordenadas físicas (e.g., latitude/longitude), várias aplicações sensíveis à localização necessitam apenas saber a localização simbólica de usuários e dispositivos. No LIS, regiões simbólicas podem ser estruturadas hierarquicamente e são representadas como qualquer área geográfica com uma semântica (ou identificação) bem definida, tal como uma sala específica, corredor, andar de um prédio, prédio, etc. Essa funcionalidade permite que as aplicações LBS obtenham a informação de localização em diferentes granularidade. Para tanto, o LIS implementa um modelo hierárquico de localizações, como aqueles discutidos nos trabalhos [25, 26]. Neste modelo (naturalmente representado como uma árvore), regiões simbólicas podem ser compostas por outras regiões simbólicas menores (aninhadas). Nesta hierarquia, os nós folhas são definidos como regiões simbólicas *atômicas*, as menores unidades de localização que podem ser reconhecidas pelo LIS. Os ancestrais de um nó folha da hierarquia de regiões representam as demais regiões simbólicas as quais um dado dispositivo pode estar implicitamente associado. Por exemplo, dada a hierarquia "PUC-Rio/Prédio RDC/5º Andar/Sala 501", o LIS pode inferir que um dado dispositivo se encontra no "Prédio RDC" caso a localização corrente desse dispositivo seja "Sala 501". O LIS oferece uma interface através da qual cada aplicação pode criar e gerenciar sua própria topologia de regiões simbólicas baseada nas regiões atômicas definidas pelo administrador do serviço.

1.3.7. Hydrogen

O projeto Hydrogen [18] propõe uma arquitetura e uma implementação de um *framework* para facilitar o desenvolvimento de aplicações móveis sensíveis ao contexto. Ao contrário de algumas outras infra-estruturas com a mesma finalidade, esta abordagem é totalmente executada no próprio dispositivo móvel (a exemplo do *ConBus* descrito na Subseção 1.3.5).

O *framework* é definido em uma arquitetura composta por três camadas: *camada de adaptadores*, *camada de gerência* e *camada de aplicação*. A Figura exibe o diagrama arquitetural da arquitetura Hydrogen.

A Camada de Adaptadores (*Adaptor Layer*) é a responsável pela obtenção de informações de contexto dos sensores. Nesta camada é possível ainda enriquecer os dados obtidos dos sensores através de algum tipo de processamento. Por exemplo, a partir de coordenadas GPS (latitude, longitude e altitude), o adaptador responsável por obter estes dados poderia ainda inferir um endereço postal, como o nome da rua, bairro e número da

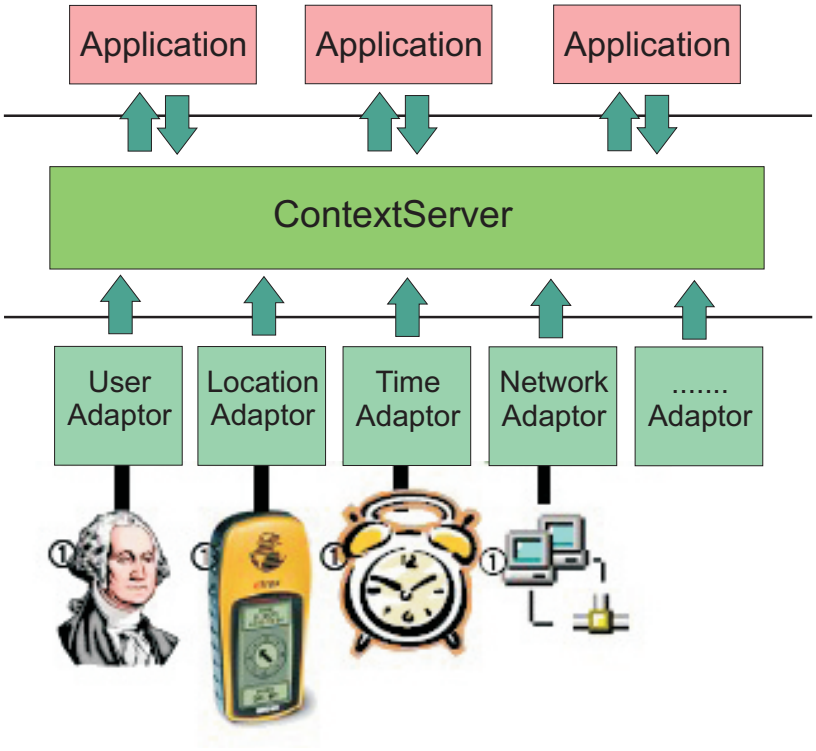

Figure 1.14. Diagrama arquitetural do Hydrogen .

localidade indicada pelas coordenadas GPS.

A Camada de Gerenciamento (*Management Layer*) tem a função de armazenar toda a informação de contexto obtida pelos sensores (mediante o intermédio dos adaptadores) sobre o ambiente atual do dispositivo móvel e disponibilizá-los às aplicações interessadas. Seu principal componente é o Servidor de Contexto (ContextServer).

O Servidor de Contexto é um objeto executável Java. É acessível através de uma porta de comunicação definida pelo usuário, onde, após ser iniciado, ficará aguardando comandos enviados pelos adaptadores e aplicações. Como foi visto anteriormente, nessa arquitetura, os adaptadores fornecem informações de contexto ao Servidor de Contexto que podem ser consultadas, de forma síncrona ou assíncrona, pelas aplicações.

Uma carência da própria arquitetura é a inexistência de módulos ou componentes na camada de gerência para realizar processamento de alto nível sobre as informações de contexto fornecidas pelos adaptadores. Ou seja, esta camada não oferece às aplicações serviços como a interpretação, agregação e classificação de contexto. De modo que praticamente o único serviço oferecido às aplicações é o armazenamento e disponibilização de contexto provenientes dos adaptadores.

Finalmente, a Camada de Aplicações (*Application Layer*) engloba, obviamente todas as aplicações sensíveis ao contexto que utilizam as informações de contexto oriundas das camadas anteriores.

Além da arquitetura de um *framework* para facilitar o desenvolvimento de aplicações móveis sensíveis ao contexto, os autores também propõem uma arquitetura para

o contexto, isto é, para representar o contexto a ser manipulado por este *framework*. Há, basicamente, dois tipos de contexto nesta arquitetura: contexto local e contexto remoto. O contexto local é, naturalmente, toda informação de contexto obtida no próprio dispositivo, enquanto o contexto remoto significa qualquer informação de contexto fornecida por outros dispositivos logicamente relacionados e que podem se comunicar com o dispositivo em questão.

Toda informação de contexto, seja local ou remota, é, na verdade, uma coleção de objetos de contexto (*ContextObject*). Tais objetos podem ser: contexto temporal (*TimeContext*), contexto do dispositivo (*DeviceContext*), contexto de rede (*NetworkContext*), contexto de localização (*LocationContext*), contexto de usuário (*UserContext*), dentre outros.

Segundo os autores, o principal benefício desta arquitetura é o fato de as três camadas do *framework* estarem localizadas no dispositivo móvel. Isto, de acordo com os autores, torna a abordagem robusta contra possíveis e frequentes desconexões de rede. Porém, isto pode ser apenas parcialmente verdadeiro, visto que se uma aplicação faz uso de contexto remoto, a robustez contra desconexões de rede poderá não acontecer se as informações remotas forem essenciais para o funcionamento adequado da aplicação em questão.

1.4. Considerações Finais

A Computação Sensível ao Contexto é atualmente uma das áreas da Computação mais promissoras para o desenvolvimento de aplicações mais adaptadas ao ambiente ubíquo e pervasivo do Século XXI. Essas aplicações, devido às suas características particulares, diferem muito das aplicações tradicionais por serem capazes de perceber as mudanças no ambiente que as envolve (ambiente computacional, físico e do usuário) e reagir de acordo com tais mudanças nesse ambiente.

Porém, como dito anteriormente, desenvolver tais aplicações não é tarefa fácil, principalmente devido à grande diversidade de informações contextuais (localização, preferências, uso de CPU e memória, nível de energia, etc.) e às inúmeras tecnologias de sensorimento disponíveis atualmente (por exemplo, vários modelos de receptores GPS ou várias APIs para a obtenção de dados sobre a agenda de compromissos dos usuários). Por isso, o desenvolvimento de infra-estruturas para provisão de contexto (*middlewares*, *frameworks*, *toolkits*, dentre outras) é imprescindível para tornar a criação de aplicações sensíveis ao contexto mais viáveis, tornando possível sua popularização.

References

[1] G. R. Mateus and A. A. F. Loureiro, *Introdução à Computação Móvel*. 11a Escola de Computação, COPPE/Sistemas, NCE/UFRJ, 1998.

[2] M. Weiser, "The computer for the twenty-first century," *Scientific American*, pp. 94–100, september 1991.

[3] S. Loke, *Context-Aware Pervasive Systems*. Boston, MA, USA: Auerbach Publications, 2006.

[4] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK: Springer-Verlag, 1999, pp. 304–307.

[5] B. N. Schilit and M. M. Theimer, "Disseminating active map information to mobile hosts," *IEEE Network, 8(5), pages 22-32,*, 1994. [Online]. Available: http://schilit.googlepages.com/ams.pdf

[6] R. Want, A. Hopper, V. Falcão, and J. Gibbons, "The active badge location system," Olivetti Research Ltd. (ORL), 24a Trumpington Street, Cambridge CB2 1QA, Tech. Rep. 92.1, 1992. [Online]. Available: citeseer.nj.nec.com/want92active.html

[7] L. Dictionary.com, "Free online dictionary of computing," 2009.

[8] J. Ryan N., Pascoe and D. Morse, "Enhanced reality fieldwork: the context-aware archaeological assistant," *Gaffney, V., Leusen, M. v. and Exxon, S. (Eds.)*, 1997.

[9] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, January 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2004.06.002

[10] N. A. Bill N. Schilit and W. R., "Context-aware computing applications," *Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, 8(5), pages 85-90, IEEE Computer Society*, 1994. [Online]. Available: http://schilit.googlepages.com/publications

[11] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," *Technical Report TR2000-381 - Dartmouth College*, 2000. [Online]. Available: http://www.cs.dartmouth.edu/reports/TR2000-381.pdf

[12] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, pp. 263–277, 2007.

[13] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1999, pp. 434–441.

[14] A. Dey, D. Salber, and G. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," 2001. [Online]. Available: http://citeseer.ist.psu.edu/dey01conceptual.html

[15] M. P. de Sá, "Conbus: Uma plataforma de middleware de integração de sensores para o desenvolvimento de aplicações móveis sensíveis ao contexto," Dissertação de mestrado, Instituto de Informática – Universidade Federal de Goiás (INF/UFG), 2010.

[16] O. Alliance., "Osgi - the dynamic module system for java. disponível em: http://www.osgi.org/main/homepage, pesquisado em 19/03/2009," 2009. [Online]. Available: http://www.osgi.org/Main/HomePage

[17] O'Reilly., "Implementing rest web services: Best practices and guidelines. disponível em: http://www.xfront.com/rest-web-services.html, pesquisado em 19/03/2009," 2009. [Online]. Available: http://www.xfront.com/REST-Web-Services.html

[18] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, "Context-awareness on mobile devices - the hydrogen approach," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 2003.

[19] V. Sacramento, M. Endler, H. K. Rubinsztejn, L. S. Lima, K. Goncalves, F. N. Nascimento, and G. A. Bueno, "Moca: A middleware for developing collaborative applications for mobile users," *IEEE Distributed Systems Online*, vol. 5, no. 10, p. 2, 2004.

[20] MoCATeam, "Moca home page," 2005, http://www.lac.inf.puc-rio.br/moca (Last visited: April 2007).

[21] F. N. da Costa Nascimento, "Um serviço para inferência de localização de dispositivos móveis baseado em redes ieee 802.11," Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2005.

[22] MoCATeam, "Context information service (cis) home page," Laboratory for Advanced Collaboration, PUC-Rio, 2005, http://www.lac.inf.puc-rio.br/moca/cis/ (Last visited: April 2007).

[23] ——, "Event-based communication interface (eci) home page," Laboratory for Advanced Collaboration, PUC-Rio, 2005, http://www.lac.inf.puc-rio.br/moca/event-service/ (Last visited: April 2007).

[24] ——, "Communication protocol (eci) home page," Laboratory for Advanced Collaboration, PUC-Rio, 2005, http://www.lac.inf.puc-rio.br/moca/event-service/ (Last visited: April 2007).

[25] A. Ranganathan, J.Al-Muhtadi, S. Chetan, R. Campbell, and D. Mikunas, "MiddleWhere: A Middleware for Location Awareness," in *Proc. of the International Middleware Conference, Toronto*, ser. LNCS, H. Jacobsen, Ed., vol. 3231, Oct. 2004, pp. 397–416.

[26] J. Roth, "Flexible positioning for location-based services," *IADIS International Journal of WWW/Internet*, vol. I, no. 2, pp. 18–32, 2003.