Capítulo 1

Computação Ubíqua: necessidades para uma Arquitetura de Software

Cristiano da Costa, Jorge Barbosa, Adenauer Yamin e Cláudio Geyer

Abstract

The ubiquitous computing (ubicomp) area presupposes a strong integration with the real world, with focus on the user and on keeping high transparency. For the development of applications in this scenario, we need an adequate software infrastructure. This chapter, therefore, presents a general model of software architecture for ubicomp. The model includes all the fundamental challenges of the area. The focus of the proposed general model is to highlight the various requirements of ubiquitous computing that had to be addressed in a software architecture. Before presenting this model, we discuss the reasons why traditional programming models alone were not enough to develop ubiquitous systems. Furthermore, we also survey the field of ubiquitous computing, presenting a general vision.

Resumo

A área de computação ubíqua (ubicomp) pressupõe uma forte integração com o mundo real, com foco no usuário e em manter alta transparência. Para o desenvolvimento de aplicações nesse cenário, é necessária uma infraestrutura de software adequada. O trabalho proposto, consequentemente, apresenta um modelo abrangente de arquitetura de software para a ubicomp. O modelo incorpora todos os desafios considerados significantes na área. O foco do modelo abrangente proposto é em destacar os vários requisitos necessários para que a computação ubíqua que devem ser endereçados por uma infraestrutura de software. Antes de apresentar esse modelo, discutimos os motivos pelo qual modelos tradicionais sozinhos não são suficientes para desenvolvimento de sistemas ubíquos. Além disso, uma visão geral da área de computação ubíqua é apresentada.

1.1. Introdução

No clássico e visionário artigo sobre computação para o século 21, Mark Weiser (1991) resume o que é esperado da computação ubíqua (também chamada de ubicomp): acesso do usuário ao ambiente computacional, de todo lugar e a todo momento, por meio de

qualquer dispositivo. A dificuldade reside em como desenvolver aplicativos que irão continuamente se adaptar ao ambiente e continuar funcionando, à medida que as pessoas se movem ou trocam de dispositivos [Costa et al. 2008] [Grimm et al. 2001]. O desenvolvimento dessa área, entretanto, ainda é limitado pelo número exíguo de linguagens e ferramentas disponíveis [Roman et al. 2002]. Além disso, muitas aplicações sensíveis ao contexto ainda estão sendo executadas em laboratórios ao invés de estarem presentes em ambientes reais do dia-a-dia [Henricksen and Indulska 2006].

Aplicações ubíquas precisam de um middleware para interoperar entre muitos dispositivos diferentes e as demandas do usuário final [Saha and Mukherjee 2003]. O objetivo é esconder a complexidade do ambiente, isolando aplicações do gerenciamento explícito de protocolos, acesso distribuído à memória, replicação de dados, falhas de comunicação, etc. Um middleware também pode resolver problemas de heterogeneidade relacionados às arquiteturas, sistemas operacionais, tecnologias de redes e até mesmo de linguagens de programação, promovendo a interoperação entre esses componentes. Por outro lado, um framework é um ambiente, composto de APIs (Interfaces de Programação com os Aplicativos), interfaces com o usuário e ferramentas, simplificando o desenvolvimento de software e o gerenciamento em um domínio específico [Bernstein 1996]. É possível utilizar framework para construir software que executa em um middleware, o qual pode ser desenvolvido utilizando frameworks existentes.

O uso de uma infraestrutura de software especificamente orientada a ubicomp pode reduzir a distância entre a visão de Weiser e o cenário atual da computação distribuída [da Costa et al. 2008]. Por isso, esse capítulo apresenta um modelo de arquitetura genérica e abrangente para a ubicomp que emprega framework e middleware. O modelo abarca todos os desafios considerados importantes na computação ubíqua (resumidos na seção 1.2). O foco da arquitetura abrangente é destacar os numerosos requisitos necessários à computação ubíqua que devem ser endereçados por uma infraestrutura de software. Antes de apresentar o modelo, é argumentado por que modelos tradicionais de desenvolvimento não se adaptam à área de ubicomp.

O capítulo está organizado em cinco seções. A seção 1.2 apresenta os principais conceitos da área de computação ubíqua, sua evolução e propõe um conjunto de desafios que devem ser endereçados na ubicomp. Na seção 1.3 é apresentada uma proposta de arquitetura de software, genérica e abrangente, para a computação ubíqua. A seção 1.4 detalha as necessidades para uma arquitetura de software endereçar cada um dos desafios da computação ubíqua. Por fim, a seção 1.5 apresenta algumas conclusões do trabalho e destaca perspectivas futuras.

1.2. Visão Geral da Computação Ubíqua

Mark Weiser criou o termo computação ubíqua e é considerado um dos fundadores da área. Ele apresentou a ubiquidade computacional como a idéia de integrar computadores de forma transparente, aprimorando o mundo real. Weiser (1991) formula uma "nova forma de pensar em computadores no mundo, uma que considera o ambiente natural do ser humano e permite que os computadores desapareçam no entorno". Computadores irão desaparecer como uma consequência da psicologia humana: quando as pessoas usam objetos sem a efetiva consciência do ato, elas focam além. Esse é um fenômeno

definido por alguns filósofos e psicólogos [Weiser 1991]: pessoas deixam de estar ciente de algo quando elas usam um objeto suficientemente bem e frequentemente. O filosofo Heidegger designa esse fenômeno Manualidade (*Vorhandenheit* no original) e Edmund Husserl denomina Horizonte (*Horizont* no original).

Heidegger faz uma análise fenomenológica da forma como as pessoas lidam com o mundo. De acordo com ele, nosso primeiro comportamento com entidades tais como ferramentas, dispositivos e sistemas no mundo é a de uso. Essas entidades, vistas do aspecto do uso, estão disponíveis e fazem parte de um modo de ser descrito por Heidegger como Manualidade [Heidegger 1996].

Edmund Husserl foi o primeiro a propor o conceito de Horizonte [Keen 1975]. Husserl era um filósofo, um dos fundadores da fenomenologia, e um matemático. O conceito refere à experiência humana como pano de fundo para tornar as experiências possíveis. Horizonte aponta para uma rede de mecanismos conhecidos focando não tanto nos objetos físicos, mas em um padrão ordenado que nós formulamos implicitamente no nosso ato de ser [Keen 1975].

Para obter a integração física dos computadores no mundo, como pano de fundo, é necessário aplicar algumas mudanças conceituais. Nessa perspectiva, Weiser também define um termo denominado virtualidade personificada (*embodied virtuality*) em oposição à ideia de realidade virtual, uma vez que computadores não podem ficar limitados aos dispositivos e softwares instalados. Além disso, é inadequado considerar a Internet ou acesso a sistemas de arquivos distribuídos como exemplos de integração transparente. Weiser aponta ainda que o poder da computação ubíqua não advém da capacidade de dispositivos específicos, mas sim da interação entre diversos dispositivos.

Analisando a visão de Weiser, Saha and Mukherjee (2003) afirmam que, apesar dos significantes desenvolvimentos no hardware, computadores ainda são máquinas que executam programas em ambientes virtuais e não são um portal para um espaço de aplicações e dados. Os autores de Want et al. (2002) concordam que muitos componentes de hardware estão prontos para a computação ubíqua, como consequência de diversas melhorias desde o artigo seminal do Weiser, incluindo redes sem fio, processadores com alto desempenho e baixo consumo de energia, melhorias nas telas, alta capacidade e dispositivos de armazenamento com pouca necessidade de energia.

Para que a computação ubíqua seja uma realidade, são necessários avanços na integração física e na interoperação espontânea, como definido por Kindberg and Fox (2002). Integração entre dispositivos e o mundo físico é crucial. À medida que os componentes se movem entre dispositivos e ambientes, eles devem alterar tanto a identidade quanto a funcionalidade se adaptando de forma a melhor interoperar.

É necessário entender e suportar as práticas das pessoas no dia-a-dia para atingir a visão proposta por Weiser, oferecendo diferentes formas de experiências interativas através de dispositivos heterogêneos conectados via rede.

1.2.1. Evolução

O advento do computador pessoal (PC) no meio dos anos 70, além de tornar o computador popular, aproximou-o das pessoas; esse representa o primeiro passo em direção à computação ubíqua [Saha and Mukherjee 2003]. Entretanto, tornar um

computador pessoal é algo oposto à visão de Weiser: o computador mantém o foco da atenção, e está então isolado da situação geral.

Computação distribuída é considerada um grande passo na evolução da ubicomp. A necessidade de trocar informações e se comunicar estimulou o desenvolvimento das redes computacionais. Sistemas distribuídos se beneficiaram da infraestrutura já existente, agindo como um conjunto de computadores interconectados que usam links de comunicação em diferentes mídias e topologias. Nesses sistemas, componentes com capacidade de processamento transferem informações usando troca de mensagem através de uma variedade de protocolos para realizar a execução de tarefas distribuídas.

Outro passo importante na evolução é a *World Wide Web* (daqui para frente referida simplesmente como Web). Com a Web, informação e comunicação tornaram-se praticamente ubíquas. O simples mecanismo de estabelecer links entre recursos é uma boa forma de integrar a informação distribuída e um potencial ponto de partida para a computação ubíqua, mesmo que a Web não esteja ainda integrada com o mundo físico [Saha and Mukherjee 2003].

O passo final na evolução é a computação móvel. Ela surge de avanços em duas áreas: redes sem fio e dispositivos portáteis. Com esses dispositivos o usuário pode acessar informação em qualquer lugar, independentemente da localização física ou mobilidade [Jing et al. 1999]. A diferença em relação à computação tradicional é que os serviços vão com a pessoa e tornam-se mais presentes, expandindo as possibilidades. Combinado com o acesso a rede, esses serviços computacionais podem transformar a computação em "uma atividade que pode ser carregada" [Lyytinen and Yoo 2002].

Uma contribuição adicional da computação móvel nessa evolução, enfatizada por Satyanarayanan (2001), é a sensibilidade à localização. Pesquisas nesse campo propõem algoritmos e técnicas para detectar a localização física, incluindo ambientes internos. Diversos aplicativos para dispositivos móveis já fazem uso dessa informação contextual na sua operação, fornecendo um comportamento sensível à localização.

As possibilidades integradas geradas pelo desenvolvimento do PC, dos sistemas distribuídos, da Web, bem como da computação móvel, definem o cenário para que a computação ubíqua se desenvolva. As principais questões envolvidas na obtenção da ubiquidade são descritas na próxima seção.

1.2.2. Desafios da Computação Ubíqua

Alguns desafios devem ser endereçados para atingir a computação ubíqua, como proposta por Weiser. A Tabela 1.1 resume os principais desafios, apresenta as áreas em que eles ganham foco e os motivos principais no escopo da ubicomp.

Heterogeneidade é um desafio derivado de sistemas distribuídos. Aplicativos devem ser capazes de executar em diferentes tipos de dispositivos, com vários sistemas operacionais e interfaces com o usuário. Software deve mascarar as diferenças de infraestrutura para o usuário e gerenciar as necessárias conversões de um ambiente para o outro. Como resultado, é necessário tratar diferenças de protocolos. Nesse cenário, não é possível recriar software específico para cada dispositivo. Consequentemente, a lógica dos aplicativos deve ser criada apenas uma vez com uma abordagem independente de dispositivo.

Outro desafio relacionado, herdado de sistemas distribuídos, é escalabilidade. Na ubicomp, um grande número de usuários, dispositivos, aplicações e comunicações são esperados em uma escala sem precedentes. Além disso, seria impraticável distribuir e instalar aplicativos explicitamente. É necessário evitar soluções centralizadas para obter uma melhor escalabilidade e prevenir gargalos. Por fim, interações mais distantes, com maior custo de comunicação, devem ser reduzidas ao mínimo necessário.

Tabela 1.1. Desafios da computação ubíqua.

Característica	Área Foco	Motivo
Heterogeneidade	Sistemas Distribuídos	Diferentes tipos de dispositivos, redes, sistemas,...
Escalabilidade	Sistemas Distribuídos	Larga escala, aumento no número de dispositivos.
Dependability e Segurança	Sistemas Distribuídos e de Missão Crítica	Evitar defeitos mais frequentes ou severos do que o aceitável.
Privacidade e Confiança	Internet e Computação Móvel	Proteger dados pessoais. Garantir a confiança dos componentes.
Interoperação Espontânea	Computação Móvel	Permitir a associação e a interação.
Mobilidade	Computação Móvel	Acesso de qualquer lugar, em qualquer tempo.
Sensibilidade ao Contexto	Computação Móvel	Perceber contexto, inferir intenção e detectar mudanças.
Gerência de Contexto	Computação Móvel	Ajustar ambiente em resposta a informação percebida.
Interação Transparente com o Usuário	Computação Ubíqua	Fundir interface do usuário com mundo real. Focar na interação.
Invisibilidade	Computação Ubíqua	Permitir que computadores desapareçam no entorno.

Algumas vezes, o sistema não consegue executar de acordo com a especificação funcional. Adicionalmente, podem ocorrer problemas relacionados com erros de especificação e de implementação. Essas situações levam a falhas. Uma falha é definida como a transição de um serviço correto para um serviço incorreto [Avizienis et al. 2004]. Um serviço correto é obtido quando o sistema implementa a função desejada. Serviço incorreto deve ser detectado e a execução restaurada para o estado correto. Evitar falhas que são mais frequentes e mais severas do que o aceitável leva a *dependability*, um conceito que integra os atributos de disponibilidade, confiabilidade, segurança crítica (*safety*), integridade e facilidade de manutenção. O termo *pervasive dependability* tem sido usado para referir a essas necessidades no escopo da ubicomp [Fetzer and Högstedt 2002].

Segurança é um conceito estritamente relacionado com a *dependability* de um sistema. Um sistema é considerado seguro se existem medidas que garantem disponibilidade, integridade e confidencialidade. Existem muitos mecanismos para fornecer segurança em sistemas distribuídos que podem também ser usados na ubicomp. Entretanto, essas ações devem ser leves, preservando tanto a espontaneidade da interação quanto as limitações de alguns dispositivos, no provimento de segurança para recursos e dados do usuário [Coulouris et al. 2005].

A privacidade dos dados desse usuário é uma questão a tratar. À medida que a ubicomp tornar-se parte da vida cotidiana, dispositivos praticamente invisíveis vão coletar informações do usuário, incluindo dados pessoais, sem nem mesmo serem

notados. Garantir formas de uso e transferência dessa informação de maneira privada poderá ser extremamente difícil.

Outro conceito associado é confiabilidade (*trust*). Em um cenário altamente heterogêneo e dinâmico, a confiança em componentes para interação deve ser medida. Uma vez que não existe uma infraestrutura fixa e nem um domínio específico, é necessário usar um sistema gerenciador de confiança para medir o que deve ser exposto aos demais componentes [Robinson et al. 2005].

Integrar componentes variados disponíveis em vários dispositivos, bem como fazer a comunicação e o entendimento entre eles possível, é um desafio identificado como interoperação espontânea. Um componente interopera espontaneamente se ele "interage com um conjunto de componentes de comunicação que podem alterar tanto a identidade quanto a funcionalidade ao longo do tempo, à medida que as circunstâncias mudam" [Kindberg and Fox 2002]. Essa espontaneidade é necessária por causa da natureza volátil da ubicomp. Os componentes estão em movimento e interagindo com um conjunto de serviços que mudam constantemente.

Outro desafio, denominado mobilidade, possibilita o acesso às aplicações e aos dados onde quer que o usuário esteja independentemente da mobilidade. Isso porque em dispositivos portáteis, tais como smartphones e notebooks, o ambiente computacional costuma acompanhar o usuário. Entretanto, mobilidade física (de equipamentos ou de usuários) não é a única opção. Mover componentes como aplicações, dados e serviços (mobilidade lógica) também é desejável. Hoje em dia, muitos desses componentes estão vinculados a dispositivos específicos, dificultando que o usuário carregue esses componentes. Aplicações devem mover-se de um dispositivo para outro, e o acesso aos dados deve ser mantido (aplicações estilo siga-me) [Augustin et al. 2002].

A computação móvel também introduziu a idéia de sensibilidade ao contexto, isto é, detectar informações que caracterizam a situação de entidades nas aplicações. Essas entidades são pessoas, lugares ou objetos que são consideradas relevantes em determinada circunstância [Dey 2001]. Para obter as informações de contexto geralmente são usados sensores ou outros dispositivos que usam computação embarcada. O emprego dessas informações relativas às entidades nas aplicações gera o que é denominado de software sensível ao contexto, algo que se torna cada dia mais comum nos *smartphones*.

O ato de uma aplicação responder às informações obtidas pelos sensores é denominado de gerência de contexto. Uma vez que é possível perceber o contexto, é necessário usar esta informação e agir proativamente. Baseado nos dados obtidos dos sensores, o sistema toma decisões, tais como a configuração de serviços de acordo com a mudança ambiental ou a manutenção do histórico de eventos passados para reiniciar serviços quando o usuário reentra nesses ambientes [Lyytinen and Yoo 2002]. O gerenciamento também pode expandir a capacidade dos dispositivos pelo uso de recursos disponíveis no contexto atual.

O projeto da Interação Humano-Computador (IHC) também é um assunto significativo. Com a ubicomp, irão existir muitas formas de interação com o usuário. Além disso, à medida que os computadores tornam-se mais "inteligentes", a intensidade e a qualidade da interação humano-computador tende a aumentar [Saha and Mukherjee 2003]. O foco na interface com o usuário, no projeto de software, tem adquirido um

novo significado desde o surgimento da computação móvel e dos novos modos de interação. A fusão de dados do usuário com o ambiente real é outra razão para o desenvolvimento da área de IHC na ubicomp. Isso faz com que a atenção seja direcionada para o conceito de interação transparente com o usuário. A idéia é preservar a atenção do usuário, evitando saturação de informações [Siewiorek 2002]. Usuários devem ser capazes de focar na tarefa que desejam realizar sem distração do sistema.

O último desafio é diretamente relacionado com a própria ubicomp. Invisibilidade está relacionada com a manutenção do foco do usuário na tarefa, não na ferramenta [Weiser 1994]. Para atingir essa visão, o software deve satisfazer a intenção do usuário, ajudando-o e não gerando obstruções. As aplicações devem aprender com os usuários e, em alguns casos, deixar esses alterarem suas preferências, interagindo de maneira quase subconsciente com o sistema [Satyanarayanan 2001].

1.3. Arquitetura Abrangente para a Computação Ubíqua

Nessa seção é proposta uma arquitetura abrangente direcionada para a ubicomp que propõe o uso de framework e middleware. O modelo considera todos os desafios considerados significantes na área da computação ubíqua. O foco da arquitetura abrangente é em destacar os diversos requisitos necessários à computação ubíqua que devem ser cobertos por uma infraestrutura de software. Antes de apresentar o modelo, são discutidos os motivos pelos quais o modelo tradicional de desenvolvimento de software não é adequado na ubicomp.

1.3.1. Implementando Aplicações Ubíquas

Um grande esforço é dedicado hoje em dia para o desenvolvimento de sistemas distribuídos. Muitas linguagens e frameworks tem sido utilizados para implementar tais sistemas. O paradigma da Programação Orientada a Objetos (POO) é o modelo de programação predominantemente utilizado. Objetos distribuídos estão se tornando cada vez mais comuns. Apesar do uso e disseminação desse modelo, alguns autores afirmam que isso não é suficiente para a computação ubíqua e um novo arcabouço de programação é necessário. Uma das principais razões para isso são os desafios apresentados na seção 1.2: os modelos de programação tradicional não endereçam usualmente todos os tópicos discutidos.

Nesse texto, modelos de programação tradicional se referem às técnicas correntemente utilizadas para implementar software. Em geral, esses modelos são utilizados para o desenvolvimento de sistemas distribuídos e são baseados em POO. No cerne desses modelos estão linguagens de programação tais como Java, C++ e C#.

Existem três limitações principais nos modelos de programação tradicionais atualmente em uso, para implementar sistemas para a computação ubíqua [Costa et al. 2008][Grimm et al. 2001]:

- Distribuição é transparente: mecanismos de comunicação utilizados, tais como objetos distribuídos, RMI (*Remote Method Invocation*) e Sistemas de Arquivos Distribuídos (SAD) escondem a localização física aos desenvolvedores. Essa transparência simplifica a programação, uma vez que tanto os recursos locais

quanto remotos podem ser utilizados praticamente da mesma forma, mas isso torna a sensibilidade e a gerência de contexto mais difíceis;

- Integração de componentes através de interfaces: todos os objetos exportam uma interface de métodos para ser utilizada por outros componentes. Isso facilita a composição entre objetos mas pressupõe um acoplamento forte, complicando a adição de novos comportamentos. Usualmente, a interface é considerada bastante estável;
- Abstração de objeto: objetos encapsulam código e dados. Manter dados dentro de objetos torna o compartilhamento de dados mais difícil. Também, os dados são tipicamente armazenados sem uma definição de formato própria.

Adicionalmente às limitações citadas, modelos de desenvolvimento tradicionais comumente são baseados em suposições estáticas: arquiteturas, aplicações, dados, sistemas operacionais, etc. Além disso, em geral, todos os recursos que serão utilizados devem ser conhecidos a priori. Para tornar a questão ainda mais complicada, interfaces de aplicativos são normalmente desenvolvidas de forma integrada com a lógica do programa. Como resultado, não é fácil criar aplicativos ubíquos, integrados de forma transparente com o ambiente usando somente modelos tradicionais e OOP.

Uma limitação importante dos modelos tradicionais de programação é a falta de suporte a mudanças no sistema. Frequentemente, intervenção manual é necessária para contemplar essas mudanças. Em Saha and Mukherjee (2003) os autores defendem que a adaptação ao ambiente é uma das características principais que diferencia a computação ubíqua da computação tradicional.

Apesar de ser necessário atacar diversas características na computação tradicional para desenvolver aplicações ubíquas, é necessário considerar o suporte a código legado, sistemas operacionais populares, uso de dados existentes e o conhecimento atual dos usuários em como utilizar os sistemas e softwares atuais [Kindberg and Fox 2002]. Por causa disso, é normal que arquiteturas para o desenvolvimento e execução de aplicações ubíquas sejam baseadas em modelos tradicionais com funcionalidades estendidas. Tipicamente, uma infraestrutura de software é construída, criando camadas de abstração para hardware, sistemas operacionais e modelos de programação tradicionais, adicionando um conjunto de novos serviços que endereçam as limitações gerais.

1.3.2. Modelo de Arquitetura

Uma arquitetura abrangente direcionada para a computação ubíqua é apresentada. Esse modelo considera todos os desafios explicados antes. Além disso, esse modelo amplia o espectro de sistemas e linguagens comumente usados. Por causa disso, métodos atualmente utilizados para comunicação remota, tolerância a falhas, alta disponibilidade, acesso à informação remota e segurança podem ser herdados.

O foco da arquitetura abrangente é em destacar vários requisitos necessários à computação ubíqua que devem estar presentes em uma infraestrutura de software. Esse modelo deve ser útil também em classificar propostas.

A Figura 1.1 apresenta o modelo de infraestrutura abrangente proposto, incluindo cada um dos desafios destacados antes e as características correspondentes que devem estar disponíveis para endereçá-las. A estrutura é então dividida

considerando o ciclo de vida de uma aplicação (tempo de projeto, tempo de carga e tempo de execução), como em Banavar et al. (2000). O tempo de projeto é quando um aplicativo é concebido, estendido ou mantido. Em tempo de carga, aplicativos são carregados a dispositivos específicos. Em tempo de execução, aplicações são executadas e utilizadas pelo usuário.

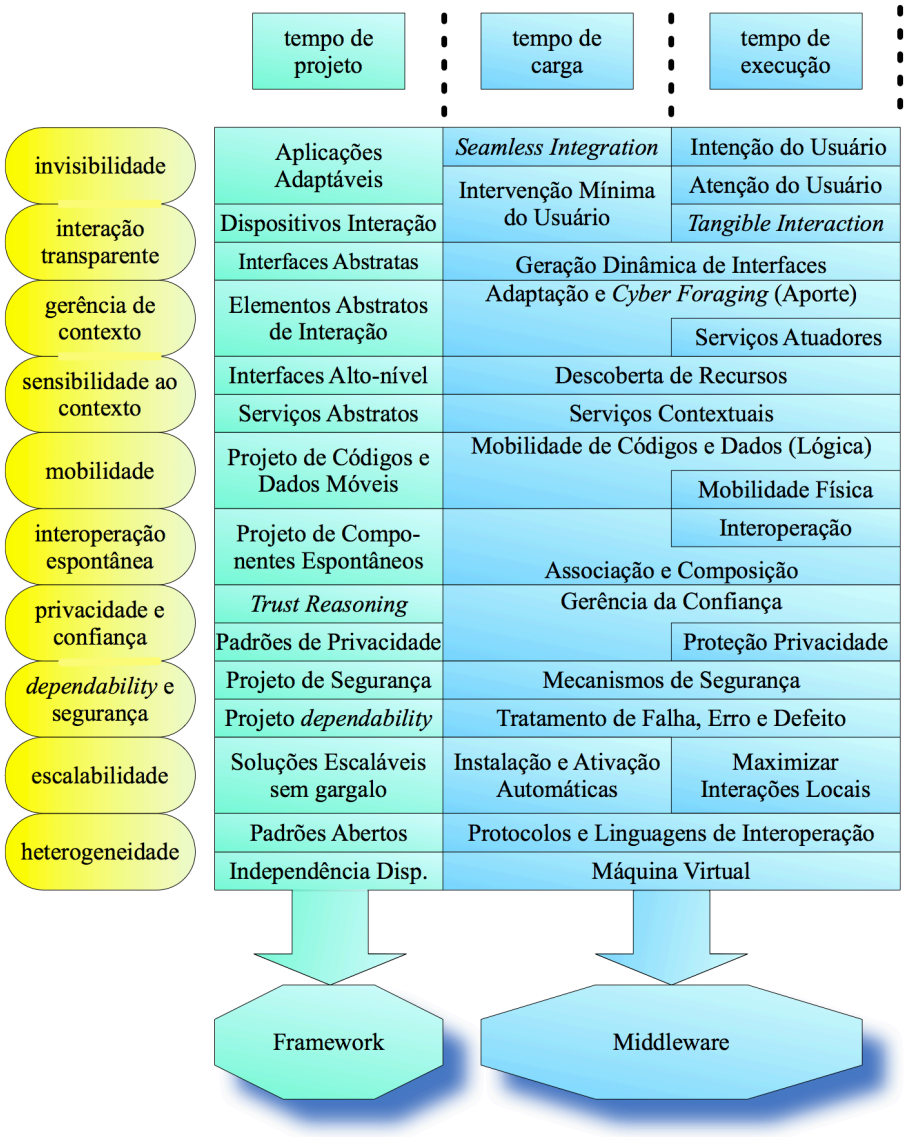

Figura 1.1. Arquitetura Abrangente

Cada linha, na Figura 1.1, apresenta um desafio (em uma caixa oval no lado mais à esquerda da figura) e, no lado direito, é possível encontrar as características essenciais que devem ser abarcadas em tempo de projeto, tempo de carga e tempo de execução respectivamente. Algumas características, como por exemplo *dependability* e segurança, e privacidade e confiança, estão mais fortemente relacionadas entre si. Por isso, são representadas sem uma linha horizontal de separação. A forte dependência também envolve sensibilidade ao contexto / gerência de contexto e interação

transparente com o usuário / invisibilidade, tornando difícil delinear uma fronteira exata entre esses conceitos.

A ordem dos desafios não implica num modelo em camadas, no qual cada camada depende dos serviços fornecidos pela camada imediatamente inferior. Serviços aparecem de baixo para cima, com os de mais baixo nível primeiro. Na parte inferior, são apresentados desafios já discutidos no âmbito de sistemas distribuídos. A figura também mostra desafios mais relacionados com computação móvel no meio e os desafios que surgem com a ubicomp no topo.

Um framework pode fornecer as abstrações necessárias à computação ubíqua em tempo de projeto. A coluna de tempo de projeto mostra todas as características necessárias nesse estágio. O mesmo se aplica ao tempo de carga e execução. Entretanto, para fornecer as características requeridas nesses estágios, é sugerido o uso de middleware.

1.4. Necessidades para uma Arquitetura de Software

Nessa seção uma discussão mais detalhada é feita das necessidades para uma arquitetura de software na ubicomp, ou seja, de cada linha do modelo de arquitetura abrangente apresentado na Figura 1.1. Uma vez que os desafios já foram descritos, o foco será nas características propostas para endereçar cada um deles.

1.4.1. Heterogeneidade

Existem vários níveis de heterogeneidade, tanto em hardware (redes, dispositivos, tamanhos de tela, autonomia da bateria, etc.) como em software (linguagens, componentes, estruturas, etc.). Para diminuir a distância entre sistemas heterogêneos, devem ser usados padrões abertos, com interfaces publicadas e mecanismos padronizados de comunicação, permitindo uma extensão ou reimplementação do sistema facilitada.

Além disso, o uso de frameworks para projetos independente de dispositivos torna possível o uso do mesmo código fonte por dispositivos diferentes, algumas vezes com pequenas alterações.

Uma das soluções bastante usadas hoje em dia para heterogeneidade é um middleware com uma API comum e com um formato binário uniformizado. Esse arquivo binário deverá executar em uma máquina virtual, que deverá estar disponível em todas as plataformas de execução. Entretanto, dependendo da capacidade dos dispositivos, não é possível sempre empregar o mesmo código binário ou esperar que o conjunto de características disponíveis não se altere. Apesar disso, o uso de máquina virtual reduz o custo da heterogeneidade porque poucas mudanças são necessárias comparadas a linguagens que geram códigos de máquina específicos.

Finalmente, é necessário focar na interoperabilidade dos componentes. Linguagens de interoperação, como XML, são bastante usadas, tornando possível representar os dados de forma padronizada e estruturada, mais portável entre diferentes aplicações. Ainda, protocolos que podem negociar serviços e recursos entre aplicações e dispositivos podem estar disponíveis, permitindo uma integração maior durante a carga e a execução.

1.4.2. Escalabilidade

Para endereçar o problema de escalabilidade, é preciso desenvolver software que considere a abundância de usuários, interações, componentes e dispositivos, evitando soluções centralizadas e gargalos. O gerenciamento e a carga de aplicações deve ser feita automaticamente durante o tempo de carga. Além disso, sempre que uma nova aplicação esteja disponível, ela deve ser automaticamente baixada e instalada, uma vez que distribuição e instalação manual de software, para cada dispositivo, seria impraticável.

Durante o tempo de execução, interações com recursos distantes, cujos custos de comunicação são maiores, devem ser reduzidas. Essa idéia, denominada de escalabilidade localizada [Satyanarayanan 2001], deve ser um objetivo da ubicomp, mesmo se ela discorda do conceito atual de transparência de rede, em que recursos locais e remotos são acessados com operações praticamente idênticas, ocultando a localização física.

1.4.3. *Dependability* e Segurança

Durante o desenvolvimento de aplicativos, verificação pode diagnosticar e remover falhas. Verificação é o processo de checar se um sistema adere a certas características. Falhas detectadas devem ser diagnosticadas, corrigidas e então o processo de verificação deve ser repetido [Avizienis et al. 2004]. Teste de software é um tipo muito usado de verificação dinâmica.

Estratégias de detecção e recuperação de falhas em uso hoje em dia, tais como uso de checkpoints, compensação, isolamento ou reconfiguração, podem ser aplicados também a ubicomp. Entretanto, existem alguns pontos a observar na ubicomp, devido à diferença de requisitos em relação à computação tradicional, uma vez que aplicativos executam em ambientes e há sempre o contexto envolvido. Ainda, dispositivos são meios de acessos a aplicativos, mas algumas falhas nos dispositivos podem não ter sido especificadas na aplicação ou no middleware. Além das falhas de dispositivos e aplicativos, devemos também considerar falhas de redes e de serviços [Chetan et al. 2005].

O projeto de segurança de um sistema ubíquo deve considerar alguns aspectos [Dourish et al. 2004]. Primeiro, ele deve ser centrado no usuário, isto é, considerar usabilidade. Usuários podem contornar mecanismos de segurança que são discordantes de práticas comuns [Bardram 2005]. Segundo, segurança depende do contexto e por causa disso o mecanismo deve estar próximo da atividade na qual ela faz sentido. Terceiro, o projeto deve ser feito de forma que os usuários entendam e gerenciem as soluções utilizadas. Somente assim o usuário pode selecionar o mecanismo adequado à necessidade de segurança em cada ação e contexto.

1.4.4. Privacidade e Confiança

Apesar de privacidade ser tipicamente um assunto de legislação, tecnologia deve ser aplicada nesse novo cenário de ubiquidade devido aos riscos do usuário expor muita informação pessoal, muitas vezes sem saber que está sendo monitorado. Além disso, um aumento é esperado na quantidade e na precisão dos dados coletados. Ainda, a proteção da privacidade é particularmente difícil em sistemas ubíquos por causa da sensibilidade de localização. Os mecanismos de sensibilidade ao contexto, particularmente os

direcionados a detectar a exata localização do usuário podem ser explorados para o propósito de *tracking*. Com esse mecanismo, é possível inferir o movimento dos usuários e também suas atividades, associando esses dados com as informações pessoais.

Durante o projeto, é possível aplicar padrões de privacidade. Esses padrões são reforçados por jurisdição e também pelo mercado, e consistem em um grupo de procedimentos que devem ser observados quando da coleta de dados [Robinson et al. 2005]. Durante a fase de execução, é possível empregar mecanismo de proteção para garantir esses padrões. Por exemplo, dados podem ser acumulados anonimamente ou deletados após um período de tempo.

Um gerenciador de confiança pode estabelecer a confiança na relação entre componentes para a troca de informações e acesso a recursos. A dificuldade reside em definir precisamente a confiabilidade de uma entidade e garantir permissão com base nessa decisão. Em alguns casos, existe pouca ou até mesmo nenhuma evidência disponível sobre a entidade em que será feita a interação e, como nas decisões diárias de confiança das pessoas, é mais uma noção subjetiva. Apesar de ser subjetiva, confiança tem outras características [Cahill et al. 2003]: não-simetria (dois componentes interagindo podem ter confiabilidade diferente entre si), específica para cada situação (dependente do contexto), dinâmica (aumenta ou diminui com o passar do tempo) e a confiança é fortemente associada com o risco (não há razão para confiar se não há risco envolvido). Por causa disso, é comum o suporte a um gerenciador de confiança que analise as entidades que irão interagir. Essa análise é feita baseada nas informações disponíveis e considerando os vários aspectos da confiança. Soluções para casos de incerteza devem ser previstas.

1.4.5. Interoperação Espontânea

O primeiro passo é projetar componentes espontâneos, ou seja, entidades que suportam a mudança frequente de parceiros de comunicação e que possam facilmente interagir com outras entidades. Em tempo de projeto, a disponibilidade de um framework pode facilitar o desenvolvimento de componentes espontâneos e fornecer uma interface genérica que irá combinar entidades específicas durante a execução. Idealmente, são empregadas linguagens de descrição uniformes para a especificação de componentes e a construção desses independentemente do contexto [Niemelã and Latvakoski 2004].

Durante a execução, componentes associam-se uns com os outros. Associação é o estabelecimento de uma relação lógica entre componentes que permitem interação; essas interações são chamadas de interoperação [Coulouris et al. 2005]. Três pontos são importantes na associação de componentes [Coulouris et al. 2005][Kindberg and Fox 2002]: escala - escolha eficiente dos componentes para associar no cenário com vários parceiros possíveis; escopo - definir a extensão em que componentes devem ser considerados e incluir todos possíveis parceiros; princípio do limite (*boundary principle*) - considerar os limites físicos (ou outros critérios) quando for definido o escopo da associação. Podem ser empregados serviços de descoberta (no modelo abrangente proposto é uma característica de sensibilidade ao contexto) como parte de solução para associação.

Interoperação depende dos modelos de comunicação empregados. Na ubicomp, são usados muitas vezes modelos baseados em sistemas de eventos ou espaços de

tuplas, devido à natureza assíncrona do primeiro ou da facilidade de desenvolvimento e persistência inerente do segundo. Ocasionalmente, ambos os modelos são usados no mesmo middleware. Além disso, é possível aplicar outras formas de comunicação para interoperação, tais como troca de mensagens, invocação remota ou sistemas de agentes.

Composição é um caso especial de associação, na qual componentes externos controlam componentes internos, em que toda interoperação passa pelo primeiro, redirecionando ou modificando a associação. Composição muitas vezes é empregada, pois facilita a adaptação e a mobilidade. Cada dispositivo pode ter um componente específico encapsulando outros e realizando as mudanças requeridas para suas interfaces específicas e capacidades. Quando um componente migra de um dispositivo para outro, ele entra no componente específico do dispositivo e continua a empregar o mesmo conjunto de operações. O processo de adaptação fica a cargo do componente externo de cada dispositivo, realizando o redirecionamento de mensagens ou de eventos que chegam após a migração de um componente interno.

1.4.6. Mobilidade

Na ubicomp, usuários trocam de dispositivos frequentemente, mas aplicações do usuário e dados devem estar sempre disponíveis. Isso significa que o ambiente deve migrar de um dispositivo para outro. Além disso, migração também ajuda a reduzir os custos de comunicação ou prevenir desconexão.

Para suportar migração de código durante a carga e execução, componentes devem ser projetados usando tecnologia móvel. Isso pode ser obtido com o uso de linguagens e sistemas compatíveis com mobilidade de código [Fuggetta et al. 1998]. Durante a execução, o middleware lida com os componentes móveis e gerencia a migração. Para atingir esse objetivo, o middleware tem de estar ciente da rede, e não tratá-la de forma transparente.

Também é necessário endereçar mobilidade de dados. Nem sempre é possível utilizar acesso a dados remotos, devido à possibilidade de desconexão ou deficiência de recursos. Nessas situações, dados podem ser movidos ou copiados para diferentes localizações, desde que seja observada a coerência de cache e demais detalhes de sincronização. Também, conversão entre formatos diferentes, para aplicações ou hardware específicos, pode ser necessária.

Além de suporte à mobilidade de código e de dados, também conhecida como mobilidade lógica, é necessário considerar a mobilidade física. À medida que as pessoas se movem, os dispositivos em uso irão alterar seus endereços de rede. Isso ocorre uma vez que a comunicação se dá com diferentes pontos de acessos, ou células no caso da rede telefônica, e necessitam do correto roteamento e endereçamento. Por exemplo, no caso de redes WiFi, os dispositivos em movimento podem receber diferentes endereços IPs à medida que se deslocam. Nesse caso, o DHCP fornece a aquisição dinâmica de endereços, permitindo que os dispositivos mantenham acesso ao serviço, independentemente da localização. Entretanto, pode ser difícil para outros componentes interoperar com esses dispositivos, uma vez que o mecanismo de roteamento IP é baseado em localização física e pode perder pacotes na mudança de endereço. Além disso, as atualizações de DNS são lentas, devido ao extensivo uso de cache.

Para suportar mobilidade física, é necessário uma estratégia de gerência de localização. Conceitualmente, essa estratégia consiste em duas operações [Adelstein et

al. 2005]: busca – operação invocada por um nodo que necessita se comunicar com dispositivos móveis; e atualização ou registro – operação realizada por um novo móvel para informar a localização corrente. Outra questão crucial é a garantia de que um nodo móvel continua conectado enquanto se move de um escopo para outro. Isso é conhecido como *handoff* e envolve os seguintes passos [Adelstein et al. 2005]: decidir quando mudar para um novo escopo, selecioná-lo, adquirir recursos, e refazer o roteamento dos pacotes para a nova localização.

1.4.7. Sensibilidade ao Contexto

Para ser ubíquo, o middleware deve usar informações e serviços relevantes disponíveis na infraestrutura computacional associada ao mesmo. Descoberta (*discovery* no original) é o componente que detecta serviços e dispositivos no contexto atual, ao passo que os sensores inferem informações significativas que podem ser usadas pelo gerenciador de contexto para decidir ações a serem tomadas.

É necessário suporte de um framework para auxiliar a implementação de aplicações sensíveis ao contexto. Duas características são fundamentais nisso [Dey 2001]: um conjunto de serviços abstratos que programadores podem utilizar para construir seus componentes e interfaces de alto nível que escondem dispositivos específicos ou detalhes de sensores do usuário.

Durante a execução, é necessário armazenar e compartilhar dados de contexto gerados pelos sensores. Tipicamente é utilizada uma representação uniforme dos dados para melhorar o acesso a dados de qualquer lugar e de várias aplicações diferentes. Para um sistema verdadeiramente ubíquo, ao invés de somente representar os dados, é necessária alguma forma de representação de conhecimento. Ontologias podem ser usadas para representação semântica explícita. Um modelo possível, entre várias soluções em desenvolvimento, é o SOUPA – *Standard Ontology for Ubiquitous and Pervasive Applications* [Chen et al. 2004], que é uma ontologia compartilhada especificamente criada para a ubicomp.

Para gerenciar a informação contextual, o middleware deve fornecer quatro categorias de serviços contextuais [Dey 2001][Adelstein et al. 2005]: assinatura e notificação de serviço – um serviço que notifica um componente da ocorrência de algum evento; consulta de contexto – um mecanismo para encontrar informações ou serviços adequados; transformação de contexto – a conversão de dados de baixo nível em informação de alto nível; síntese de contexto– a agregação de informações de contexto para gerar um contexto mais preciso ou detalhado. Além desses serviços, é recomendado um serviço de descoberta dinâmica, para auxiliar a busca de entidades disponíveis no contexto atual.

1.4.8. Gerência de Contexto

A detecção de alterações no contexto pode afetar o comportamento do sistema. Essa mudança pode ser feita adaptando o sistema a novas condições ou aumentando a quantidade de recursos disponíveis para compensar a falta de alguma característica. Outra possibilidade é alterar o contexto pelo uso de atuadores, ou seja, dispositivos controlados por software que afetam o mundo real. Um atuador pode ativar um dispositivo, alterar uma condição física (com a temperatura de um ambiente) ou executar uma ação lógica (carregar um código, alterar a parametrização, mover

componentes, etc.). Para suportar esse gerenciamento, são necessários elementos abstratos de interação em tempo de projeto. Esses elementos também podem ser empregados durante a execução, de acordo com o contexto.

As duas características mais importantes da gerência de contexto são adaptação e aporte (*cyber foraging* no original) [Satyanarayanan 2001]. Adaptação consiste em ajustar aspectos das aplicações a mudanças no ambiente. Um caso especial de adaptação é aporte. Dispositivos móveis geralmente tem capacidade limitada, como poder de processamento, memória e autonomia de bateria. Com essas limitações, é muitas vezes difícil satisfazer as necessidades computacionais dos usuários. Para minimizar esse problema, é possível usar máquinas próximas como servidores adicionais para processamento e armazenamento de dados, assim aumentando a capacidade dos dispositivos móveis [Satyanarayanan 2001]. Aporte significa compartilhar e dividir o código e os dados entre servidores e dispositivos móveis, o que pode ser automaticamente feito pelo middleware durante a carga ou execução. Alternativamente, o processo de aporte pode ser iniciado pelo usuário, por exemplo quando antecipar mudanças na conectividade ou troca de dispositivo.

1.4.9. Interação Transparente

É necessário projetar aplicações neutras em termos de dispositivos. Para conseguir isso, durante o tempo de projeto, é necessário definir interfaces abstratas com o usuário e prever diferentes tipos de interações, de forma que a decisão de qual interface será utilizada fica postergada para tempo de execução. Outra opção é gerar dinamicamente a interface durante a execução, baseada em definições abstratas, características específicas do dispositivo e informação contextual. Essa opção requer menos esforço durante o projeto e tende a consumir mais poder de processamento e latência de comunicação durante a execução. Entretanto, facilita o uso de dados contextuais.

A geração de interfaces adequadas para cada tipo específico de dispositivo é uma das características importantes para que se obtenha interação transparente. Essas interfaces devem considerar a forma mais natural de interação para cada dispositivo específico e também informações contextuais e comportamento do usuário (preferências, necessidades, história, etc.) [Canny 2006]. Por exemplo, reconhecimento de voz é uma das melhores interfaces para telefones celulares por causa das telas de tamanho pequeno, teclados reduzidos e otimização para comunicação por voz [Canny 2006].

Um conceito mais amplo envolve não somente focar na interface humano-computador dos dispositivos, mas no projeto da própria interação física. Essa idéia leva ao que se tem denominado de interação tangível e sua aplicação no escopo da ubicomp [Holmquist 2004]. A proposta é criar uma experiência de interação mais rica, acoplando informações digitais a artefatos físicos, usando o corpo humano como uma interface e combinando objetos reais e dispositivos com computadores em espaços interativos [Hornecker 2005]. O desafio consiste em criar interfaces integradas de forma transparente com o mundo real e que considerem a experiência humana social, pessoal e emocional [Ross and Keyson 2007]. Finalmente, para atingir uma transparência de fato, pessoas devem estar aptas a focar na tarefa que desejam realizar intuitivamente e ficar minimamente envolvidas com questões de gerenciamento do sistema computacional.

1.4.10. Invisibilidade

O primeiro passo para obtermos sistemas invisíveis é projetar aplicativos adaptáveis. É necessário um suporte de framework para facilitar o desenvolvimento, seguindo os objetivos de ocultar a computação e manter o foco do usuário na tarefa. Em tempo de execução, deve ser buscado um uso ininterrupto do sistema, com mínima intervenção do usuário. Por exemplo, períodos de desconexões podem ocorrer em dispositivos móveis e o sistema pode tentar mascar esse evento, mantendo serviços e ainda satisfazendo as necessidades do usuário, talvez com alguma degradação.

Uma característica da invisibilidade é o que tem sido chamado de *seamless integration* (integração inconsútil), ou seja, a associação e cooperação transparente de vários componentes. A idéia de componentes que interoperam entre si de maneira inconsútil exigem muito suporte do middleware e desenvolvimento cuidadoso de cada elemento do sistema, considerando diversos aspectos apresentados em outras camadas da arquitetura proposta.

Para ser invisível durante a execução, um sistema deve agir de forma não obstrusiva, atingindo as expectativas do usuário. O sistema não deve somente responder a ações iniciadas pelos usuários, mas também antecipar as necessidades destes, de forma não intrusiva, capturando a intenção. Preservar a atenção do usuário é outra característica que deve ser considerada, já que é um dos recursos mais importantes no sistema [Garlan et al 2002]. Invisibilidade é a característica mais difícil de ser obtida em um sistema de ubicomp.

1.5. Conclusão e Perspectivas Futuras

O desenvolvimento de software ubíquo ainda é limitado pela falta de infraestrutura de software cobrindo diversas necessidades da ubicomp. A principal razão por essa ausência é a complexidade de considerar tópicos de pesquisa em aberto tão diferentes em um único projeto. Entretanto, muitos projetos hoje em dia têm fornecido soluções isoladas para questões específicas.

Nesse capítulo foram apresentadas as principais questões relacionadas com o desenvolvimento de software ubíquo. Primeiro, os principais conceitos da área foram apresentados, focando nos desafios encontrados. Segundo, foram destacadas as dificuldades em construir software ubíquo. Em seguida, foram apresentadas as principais características e técnicas para endereçar cada desafio apresentado.

Estamos presenciando o início da era da computação ubíqua. Apesar de existir uma grande evolução em linguagens e ferramentas para o desenvolvimento de software desde o advento do PC, quando o foco são os requisitos da ubicomp, ainda se observa que alguns aspectos no desenvolvimento estão nos primeiros passos. São necessários middleware e framework para acelerar o processo de desenvolvimento de software nesse cenário.

Um problema adicional do cenário da ubicomp é como a infraestrutura pode permitir que o usuário acesse o ambiente computacional dele onde estiver e independentemente do deslocamento. A promessa do "todo tempo, de todo o lugar" que seguidamente é apresentada junto com a ubicomp ainda é difícil de ser obtida. Outro problema relacionado é como usar dados e aplicativos fortemente integrados com o

mundo real. Essas questões envolvem o endereçamento conjunto de diversos desafios aqui apresentados.

Esperamos que esse capítulo possa contribuir para o avanço do desenvolvimento da computação ubíqua. Além disso, consideramos que para alcançar a visão seminal do Weiser de ubicomp, futuros sistemas devem integrar-se transparentemente com o ambiente, concomitantemente tratando várias questões relacionadas com os desafios discutidos nesse trabalho.

1.6. Referências

Adelstein, F. et al., Fundamentals of Mobile and Pervasive Computing, McGraw-Hill, 2005.

Augustin, I. et al., "Towards Taxonomy for Mobile Applications with Adaptive Behavior," Proc. 20th Int'l Symp. Parallel and Distributed Computing and Networking (PDCN 02), ACTA Press, pp. 224-228, 2002.

Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C.; , "Basic concepts and taxonomy of dependable and secure computing," Dependable and Secure Computing, IEEE Transactions on , vol.1, no.1, pp. 11- 33, Jan.-March 2004. doi: 10.1109/TDSC.2004.2

Banavar, G. et al., "Challenges: an Application Model for Pervasive Computing," Proc. 6th Int'l Conf. Mobile Computing and Networking (MOBICOM 00), ACM Press, pp. 266–274, 2000.

Bardram, J., "The Trouble with Login: on usability and computer society security in ubiquitous computing," Personal and Ubiquitous Computing, vol.9, no.6, pp. 357-367, Nov. 2005.

Bernstein, P. "Middleware: a model for distributed system services," Communications of the ACM, vol.39, no.2, pp. 86-98, Feb. 1996.

Cahill V. et al., "Using Trust for Secure Collaboration in Uncertain Environments," IEEE Pervasive Computing, vol. 2, no. 3, pp. 52–61, 2003.

Canny, J., The Future of Human-Computer Interaction, ACM Queue, vol.4, no.6, pp. 24-32, July 2006.

Chen, H.; Perich, F.; Finin, T.; Joshi, A.; , "SOUPA: standard ontology for ubiquitous and pervasive applications," Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on , vol., no., pp. 258- 267, 22-26 Aug. 2004. doi: 10.1109/MOBIQ.2004.1331732

Chetan, S.; Ranganathan, A.; Campbell, R., "Towards Fault tolerant Pervasive Computing," IEEE Technology and Society Magazine, vol.24, no.1, pp.38-44, 2005.

Coulouris, G. et al., "Mobile and Ubiquitous Computing," Distributed Systems: Concepts and Design, 4th ed., Addison-Wesley, pp. 657–719, 2005.

da Costa, C.A.; Yamin, A.C.; Geyer, C.F.R.; , "Toward a General Software Infrastructure for Ubiquitous Computing," Pervasive Computing, IEEE , vol.7, no.1, pp.64-73, Jan.-March 2008. doi: 10.1109/MPRV.2008.21

Dey, A., “Understanding and Using Context,” Personal and Ubiquitous Computing, vol. 5, no. 1, ,pp. 4–7, 2001.

Dourish P. et al., “Security in the Wild: User Strategies for Managing Security as an Everyday, Practical Problem,” Personal and Ubiquitous Computing, vol.8, no.6, pp. 391–401, 2004.

Fetzer, C.; Högstedt, K. “Challenges in Making Pervasive Systems Dependable,” Future Directions in Distributed Computing, A. Schiper et al., eds., Springer, pp.186–190, 2002.

Fuggetta, A.; Picco, G.P.; Vigna, G.; , "Understanding code mobility," Software Engineering, IEEE Transactions on, vol.24, no.5, pp.342-361, May 1998. doi: 10.1109/32.685258

Garlan, D. et al., “Project Aura: Toward Distraction-Free Pervasive Computing,” IEEE Pervasive Computing, vol. 1, no. 3, pp. 22–31, 2002.

Grimm, R.; Davis, J.; Hendrickson, B.; Lemar, E.; MacBeth, A.; Swanson, S.; Anderson, T.; Bershad, B.; Borriello, G.; Gribble, S.; Wetherall, D.; , "Systems directions for pervasive computing," Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on , vol., no., pp. 147- 151, 20-22 May 2001. doi: 10.1109/HOTOS.2001.990075

Heidegger, M., Being and time: a translation of Sein and Zeit, State University of New York, New York, 1996.

Henricksen, K.; Indulska, J.; , " Developing Context-aware Pervasive Computing Applications: models and approach," Pervasive and Mobile Computing, vol.2, no.2, pp. 37-64, Feb. 2006.

Holmquist, L.; Schmidt, A.; Ullmer, B., "Tangible Interfaces in perspective," Personal and Ubiquitous Computing, vol.8, no.5, pp. 291-293, May 2004.

Hornecker, E. “A Design Theme for Tangible Interaction: Embodied Facilitation,” Proc. 9th European Conf. Computer Supported Cooperative Work (ECSCW 05), Kluwer, pp. 23–43, 2005.

Jing, J.; Helal, A.; Elmagarmid, A., "Client-server Computing in Mobile Environments," ACM Computing Surveys, vol.31, no.2, pp. 117-157, June 1999.

Keen, E., A primer in phenomenological psychology, Holt, Rinehart and Winston, New York, 1975.

Kindberg, T.; Fox, A.; , "System software for ubiquitous computing," Pervasive Computing, IEEE , vol.1, no.1, pp. 70- 81, Jan-Mar 2002. doi: 10.1109/MPRV.2002.993146

Lyytinen, K.; Yoo, Y., "Issues and Challenges in Ubiquitous Computing," Communications of the ACM, vol.45, no.12,pp. 63-65, Jan. 2002.

Niemelä, E.; Latvakoski, J., “Survey of Requirements and Solutions for Ubiquitous Software,” Proc. Mobile Ubiquitous Computing Conf., ACM Press, pp. 71–78, 2004.

Robinson, P. et al., “Some Research Challenges in Pervasive Computing,” Privacy, Security and Trust within the Context of Pervasive Computing, P. Robinson et al., eds., Springer, pp. 1–16, 2005.

Roman, M.; Hess, C.; Cerqueira, R.; Ranganathan, A.; Campbell, R.H.; Nahrstedt, K.; , "A middleware infrastructure for active spaces," Pervasive Computing, IEEE , vol.1, no.4, pp. 74- 83, Oct-Dec 2002. doi: 10.1109/MPRV.2002.1158281

Ross, P.; Keyson, D., "The case of sculpting atmospheres: towards design principles for expressive tangible interaction in control of ambient systems," Personal and Ubiquitous Computing, vol.11, no.2, pp. 69-79, Feb. 2007.

Saha, D.; Mukherjee, A.; , "Pervasive computing: a paradigm for the 21st century," Computer , vol.36, no.3, pp. 25- 31, Mar 2003. doi: 10.1109/MC.2003.1185214

Satyanarayanan, M., “Pervasive Computing: Vision and Challenges,” IEEE Personal Communications, vol.8, no.4, 2001, pp. 10–17.

Siewiorek, D., "New frontiers of application design," Pervasive Computing, IEEE, vol.1, no.1, pp. 79-82, Jan. 2002.

Want, R.; Pering, T.; Borriello, G.; Farkas, K.I.; , "Disappearing hardware [ubiquitous computing]," Pervasive Computing, IEEE , vol.1, no.1, pp. 36- 47, Jan-Mar 2002. doi: 10.1109/MPRV.2002.993143

Weiser, M., "The world is not a desktop," ACM Interactions, vol.1, no.1, pp. 7-8, Jan. 1994.

Weiser, M.; , "The computer for the 21st Century," Scientifc American, vol.265, no.3, pp.94-104, March 1991.