Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

UbiquitOS – Uma proposta de arquitetura de middleware para a adaptabilidade de serviços em sistemas de computação ubíqua

Alexandre Rodrigues Gomes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Ricardo Pezzuol Jacobi

Brasília
2007

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenadora: Profª. Drª. Alba Cristina M. de Melo

Banca examinadora composta por:

Prof. Dr. Ricardo Pezzuol Jacobi (Orientador) – CIC/UnB
Profª. Drª. Célia Ghedini Ralha – CIC/UnB
Prof. Dr. Cláudio Fernando Resin Geyer – Instituto de Informática/UFRGS

CIP – Catalogação Internacional na Publicação

Alexandre Rodrigues Gomes.

UbiquitOS – Uma proposta de arquitetura de middleware para a adaptabilidade de serviços em sistemas de computação ubíqua/ Alexandre Rodrigues Gomes. Brasília : UnB, 2007.
100 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2007.

1. computação ubíqua, 2. computação móvel, 3. arquitetura de software, 4. adaptabilidade de serviços

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

UbiquitOS – Uma proposta de arquitetura de middleware para a adaptabilidade de serviços em sistemas de computação ubíqua

Alexandre Rodrigues Gomes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Ricardo Pezzuol Jacobi (Orientador)
CIC/UnB

Profª. Drª. Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Cláudio Fernando Resin Geyer
Instituto de Informática/UFRGS

Profª. Drª. Alba Cristina M. de Melo
Coordenadora do Mestrado em Informática

Brasília, 7 de março de 2007

Dedicatória

Tércio Gomes, meu pai. Que Deus o tenha.

Agradecimentos

Seletos são os mestrandos com oportunidade de contar com uma comissão mista para orientação de suas atividades. Assim, agradeço aos professores Alba Melo, Carla Castanho, Célia Ghedini, Ricardo Jacobi, Jacir Bordin e Jorge Fernandes que, em algum dos muitos encontros organizados, marcaram suas passagens por este périplo, com dicas, broncas, idéias e sugestões.

Aos funcionários do CIC, refúgios para minhas lamentações (né Rosa? né Cida?), muito grato!

Aos amigos, valeu boi! Será que eu lembro o nome de todos? Pacote, Lorena, Nerso, Castanhede, Léo, Tânia, JG, Nardelli, Daniela, Marcelinho, Rodolfo, Roger, Robertinha, SEGFAULT, Core Dump e o cara nerd de Teoria da Computação. Sejamos resilientes agora, né Daniela?

Galera da SEA, esses sim foram brótheres.

Namorada, seguidora de Jó, santa paciência. Obrigado pela compreensão.

Mamãe, Xozo, Dudu e Brasão, amo vocês.

Meu pai, muitas saudades.

Deus, obrigado.

Resumo

Ubiquitious Computing, Pervasive Computing, Invisible Computing e Calm Technology são todos termos que definem uma nova linha de pesquisa tecnológica que se baseia na intensa pulverização da informática no mundo real buscando uma ampla e transparente integração entre objetos, dispositivos e serviços para a criação dos chamados *smart-spaces*. Nestes cenários, novos e antigos problemas computacionais disputam a preferência da comunidade de pesquisa. Dentre eles, a adaptabilidade de serviços, apesar de nem sempre tratada explicitamente, é de suma importância quando se trata da essência volátil destes ambientes, nos quais predomina a constante variação da disponibilidade dos recursos computacionais existentes. Como meio de otimizar a utilização desta potencial malha de recursos de um *smart-space*, propõe-se uma arquitetura para facilitar a adaptação de serviços. Essa arquitetura, implementada como um *middlewares* do *smart-space*, viabiliza o compartilhamento de serviços entre dispositivos, evitando, assim, o desperdício de recursos do ambiente e permitindo, consequentemente, melhor satisfação e comodidade aos usuário do sistema. Com isso, a medida em que dispositivos entram na área de cobertura do *smart-space*, os serviços por eles providos são disponibilizados, através de uma interface comum, a todos os demais elementos da comunidade, possibilitando assim que um dispositivo utilize os serviços de outro como alternativa a sua própria implementação daquele serviço. Ao se afastar do raio de alcance do *smart-space*, os serviços compartilhados pelo dispositivo são automaticamente indisponibilizados.

Palavras-chave: computação ubíqua, computação móvel, arquitetura de software, adaptabilidade de serviços

Abstract

Ubiquitious Computing, Pervasive Computing, Invisible Computing and Calm Technology are all about a new research area based on a huge integration between objects, devices and services for the composition of places called smart-spaces. In this scenarios, the research community works on new and old computing problems. Among them, service adaptability is so relevant when we remember that resources, in such environments, comes and goes. So, to allow services to adapt to these resource availability variations, this thesis propose an architecture to implement service adaptability in middlewares target to ubiquitous computing smart-space systems. Based on this architecture, when a device comes into the smart-space covered area, its services are shared with every other device in the community. So, it is possible for a device to use other's service instead of its own implementation.

Keywords: ubiquitous computing, mobile computing, software architecture, service adaptability

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"Ubiquitous Computing is fundamentally characterized by the connection of things in the world with computation."

Mark Weiser, 1991

A computação ubíqua, *ubicomp*, define uma realidade na qual o computador se integra, em um modelo de alta mobilidade, a ambientes físicos para a constituição de espaços inteligentes, *smart-spaces*, nos quais atividades do dia a dia são facilitadas com recursos e serviços computacionais [6, 7, 8, 9, 10]. Nestes cenários, é mister a manutenção de premissas básicas que caracterizam a *ubicomp*, como a invisibilidade, que garante que toda orquestração entre serviços e dispositivos do meio seja regida sem a intervenção, e mesmo a ciência, do usuário, e a pró-atividade que, dentro das possibilidades de um conjunto de informações de contexto do ambiente, antecipa as necessidades do usuário, surpreendendo-o com a prestação otimizada de serviços.

Como novo paradigma computacional, a *ubicomp* herda de arquiteturas predecessoras problemas ainda não solucionados, ao passo que também propõe desafios até então não explorados pela comunidade científica. Como tentativa de busca por respostas às questões ainda em aberto, diversos projetos de computação ubíqua foram desenvolvidos nos últimos anos, cada qual com objetivos e circunstâncias próprios. Conforme analisado, nenhum projeto possui a pretensão de suprir a todas as carências da *ubicomp*. Suas propostas consolidam-se com foco em assuntos altamente especializados. Desta forma, observa-se, na comunidade científica, a ausência de uma meta coletiva buscada por todos. O observado, outrossim, é o tratamento de cenários particulares, através dos quais se busca a demonstração das potencialidades e oportunidades da computação ubíqua.

Neste âmbito, alguns projetos de maior destaque na literatura específica foram explorados e neles percebeu-se certo comportamento, não explicitamente tratado,

mas comum em sua maioria. A possibilidade de compartilhamento de serviços entre dispositivos não é pesquisa fim de nenhum dos projetos analisados, mas compreende no mecanismo meio para viabilização dos objetivos de cada proposta. Desta observação, então, a possibilidade de um dispositivo utilizar serviços de outro, chamada de adaptação de serviço, foi extraída como denominador comum destes projetos de destaque e utilizada como foco do presente trabalho.

Em [11], Demeure contextualiza a adaptação de serviços em meio a outros conceitos, dentre os quais, destaca-se a adaptabilidade de serviços como propriedade viabilizadora da adaptação. Assim sendo, direcionou-se a pesquisa na adaptabilidade de serviços, característica básica, requerida para a constituição de *smart-spaces* com suporte à adaptação de serviços e, consequentemente, ao compartilhamento de serviços entre dispositivos, cujo valor foi heuristicamente observado naqueles projetos.

Para demonstrar a importância da adaptação de serviços e, consequentemente, da adaptabilidade de serviços, foram apresentados alguns cenários que apontam potencial desperdício de recursos computacionais em *smart-spaces* quando de sua ausência.

É proposta então uma arquitetura para a composição de *middlewares* de computação ubíqua, chamada de UbiquitOS, sobre a qual é implementada a adaptabilidade de serviços.

A presente dissertação de mestrado está organizada da seguinte maneira. O capítulo 2 apresenta os conceitos básicos da computação ubíqua e finaliza com a descrição das propriedades comumente esperadas para sua configuração. O capítulo 3 traz uma lista das principais iniciativas acadêmicas e comerciais que buscam demonstrar, dentro de seus respectivos interesses, as oportunidades emergentes da *ubicomp*, de onde se observa a abordagem indireta e constante da adaptação de serviços. No capítulo 4 são descritos conceitos satélites à adaptação de serviços, como a adaptabilidade de serviços, e é feita uma análise de sua presença em cada um dos projetos vistos no capítulo anterior. Na sequência, capítulo 5, a importância da adaptabilidade de serviços para sistemas de computação ubíqua é justificada e uma proposta para sua implementação é sugerida. O capítulo 6 descreve uma arquitetura modular e multi-plataforma capaz de dar suporte à proposta do capítulo anterior e conclui com os resultados coletados, na forma de contribuições científicas. Para finalizar, são apresentados, no capítulo 7, a análise final da pesquisa, os impactos positivos e negativos da proposta e uma lista de potenciais trabalhos futuros para a continuidade desta linha de pesquisa.

Capítulo 2

Computação Ubíqua

"The most profound technologies are those that disappear."
Mark Weiser, 1991

Este capítulo introduz os conceitos básicos da computação ubíqua, apresentando cenários que a descrevem e as propriedades que a caracterizam.

2.1 Apresentação

Inspirado por cientistas sociais, filósofos e antropólogos, em 1988, Mark Weiser e demais colaboradores dos laboratórios do Xerox PARC [12] sintetizaram o que a computação deveria se tornar no futuro: algo invisível [7]. Esta idéia foi formalizada em seu artigo de 1991 [13] quando a comunidade acadêmica viu nascer uma linha de pesquisa até então inexplorada, a computação ubíqua (*Ubiquitious Computing*), ou simplesmente *ubicomp*.

Neste novo contexto, a computação é imersa no cotidiano da sociedade de tal forma a prover a colaboração transparente entre dispositivos do meio para o fornecimento de serviços a seus usuários e proporcionar um novo cenário de simbiose entre a vida social e tecnológica.

Weiser, considerado desde então o pai deste novo modelo computacional, utilizou a escrita moderna para mostrar como uma tecnologia se torna ubíqua [8]. Outra paralelo possível de ser traçado para a demonstração de tecnologias que tornaram-se ubíquas é a evolução dos motores genéricos.

Há alguns milhares de anos atrás, as antigas civilizações da Suméria, Babilônia, Assíria e Pérsia criaram um mecanismo para representar pensamentos na forma de símbolos abstratos; nascia a escrita moderna. Aquilo que durante muito tempo foi acessível exclusivamente aos mais conceituados especialistas das letras

hoje está integralmente imerso em nosso cotidiano e imperceptivelmente é uma tecnologia consumida em larga escala.

Motores de combustão interna foram utilizados em cenários rurais por anos. Projetados como máquinas de propósito genérico, em um passado não muito distante, viam-se cidadãos do campo adaptarem diariamente suas máquinas geradoras de torque para diferentes fins, como a extração de água em cisternas, geração de energia elétrica ou a trituração de alimentos para o rebanho pecuário. Após a pulverização do fornecimento de energia elétrica pelo país e a democratização do acesso a eletrodomésticos, a figura dos *motores genéricos* de outrora não mais existe. Para extração de água de cisternas, não se usa mais "motores", usa-se bombas d'água. Para trituração de alimentos, usa-se liquidificadores, e assim por diante. Com o tempo, o motor foi aparentemente substiuído por pequenos dispositivos de aplicabilidade específica. Entretanto, essa aparente substituição é equivocada pois, apesar de não mais utilizado em cenários genéricos, o motor foi multiplicado e embarcado em cada eletrodoméstico que se dizia tê-lo substituído. Ou seja, o motor que antes era único e genérico, é agora múltiplo e especializado e, aos olhos do usuário final, o motor desapareceu, ou seja, o motor tornou-se invível mas, na verdade, o motor apenas se tornou uma tecnologia ubíqua.

A concretização do pensamento de Weiser, e consequentemente a ubiquidade da informática, terá atingido sua plenitude no dia em que a computação for aplicada com a mesma naturalidade que hoje é utilizada a língua escrita e os motores, agora elétricos e embarcados, para realização de atividades do cotidiano. Ou seja, a comunidade terá alcançado a era da computação ubíqua quando o computador deixar de ser peça única e de uso genérico e multiplicar-se para usos especializados.

Conforme defendido por Weiser em seu artigo entitulado *The World is not a Desktop* [9], de 1993, o computador é hoje uma interface de comunicação com um mundo cibernético e, como todas as interfaces de sucesso, ele tende a desaparecer. Por exemplo, os óculos são uma interface entre o ser humano e o mundo mas, o ser humano não concentra suas atenções nos óculos, mas no mundo. Uma pessoa cega, ao utilizar sua bengala, percebe o mundo ao seu redor, e não a bengala. Com algum esforço, é possível identicarmos outras interfaces que se tornaram de uso inconsciente.

Também conhecida como computação invisível [10], Mark Weiser define a computação ubíqua como sendo o terceiro grande paradigma computacional, precedido pelo império dos *mainframes* e pela onda da computação pessoal [6]. Marcante no passado e hoje presente em cenários específicos, os computadores

Era Computacional	Características
Mainframes	Uma máquina para vários usuários
Computação Pessoal	Uma máquina para cada usuário
Computação Ubíqua	Várias máquinas para cada usuário

Tabela 2.1: Eras computacionais propostas por Mark Weiser

de grande porte foram definidos sobre uma arquitetura na qual poucas máquinas de imenso poder de processamento são compartilhadas simultaneamente por inúmeros usuários. Hoje, com o domínio dos micro-computadores, observamos a máquina como peça íntima de uso pessoal e exclusivo. Na computação ubíqua, veremos a tecnologia discretamente nos envolvendo e agindo nos bastidores de nossas vidas, formando uma malha de dispositivos inteligentes em torno de cada indivíduo, sem contudo impactar profundamente nosso modo de viver. A tabela 2.1 resume as 3 eras computacionais propostas por Weiser.

Ainda que muitas vezes tratadas sem distinção, a computação pervasiva (*Pervasive Computing*) e a computação ubíqua referem-se a segmentos distintos de pesquisa e desenvolvimento tecnológico [1]. A computação ubíqua integra os avanços da computação móvel e da computação pervasiva. Enquanto a primeira objetiva-se no incremento das possibilidades de locomoção de serviços entre ambientes, o segundo conceito trata da capacidade de dispositivos computacionais serem embutidos no universo físico para a obtenção de informações do meio para a construção dinâmica de novos modelos computacionais. Em síntese, a computação pervasiva busca maior integração entre a computação e o ambiente físico no qual ela está imersa. A figura 2.1, retirada de [1], ilustra estas dimensões da computação ubíqua, relacionando computação ubíqua, computação pervasiva e computação móvel.

Neste sentido, o grande desafio da *ubicomp* é a constante busca pela fusão da mobilidade da computação móvel com a capacidade de interação com o meio agregada pela computação pervasiva, conforme ilustrado na figura 2.2. Em um contexto prático, na computação ubíqua o computador, independente de sua forma, desloca-se com o usuário e comunica-se com os recursos computacionais do ambiente em que se encontra para configuração dinâmica de novos serviços de tal forma que as necessidades do usuário sejam satisfeitas de maneira natural e transparente.

Por definição, a computação ubíqua opõe-se diretamente ao conceito de realidade virtual. Por este, o mundo real é imergido em cenários imaginários gerados computacionalmente. Pela ubiquidade, toda tecnologia computacional deve ser

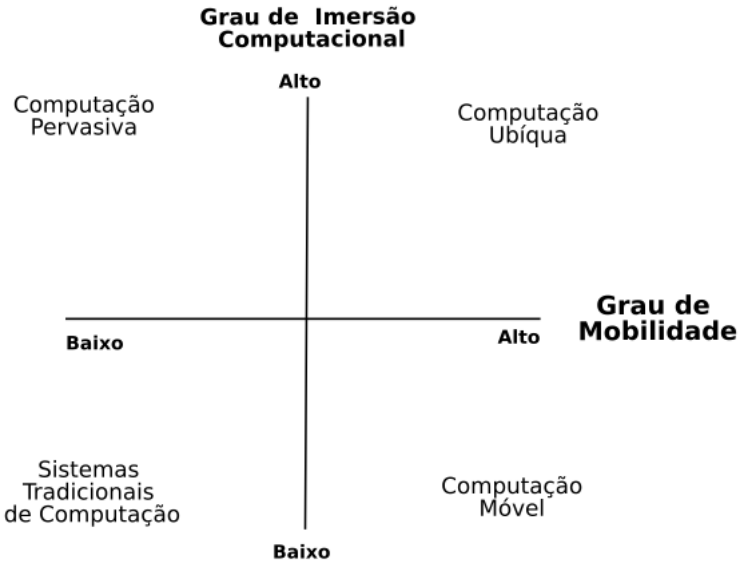

Figura 2.1: Dimensões da Computação Ubíqua [1]

Figura 2.2: Computação Ubíqua combina características das Computações Móvel e Pervasiva

naturalmente incorporada ao mundo real.

A multimídia também é um tópico que se distingue em absoluto da ubiquidade, uma vez que a primeira tem como foco a captura da atenção de seus usuários, através de cores, sons e demais artifícios ergonômicos, e a segunda preza pela discrição e *invisibilidade*.

Ilustremos estes conceitos com a adaptação de um cenário de aplicação da computação ubíqua, citado por Satyanarayanan em [14]:

> *Luciana está no portão 5 do Aeroporto Internacional de Brasília a espera de seu vôo para Patos de Minas, MG. Neste período, pelo laptop, ela revisa sua dissertação de mestrado pois necessita enviá-la por e-mail, através da rede wireless do aeroporto, ao seu orientador para correções. Entretanto, devido à limitação da largura de banda disponível naquele instante, causada pela alta taxa de utilização da rede no perímetro do portão 5, o envio imediato do documento de Luciana, que contém vários megabytes de texto, imagens e planilhas, é inviabilizado, devido à iminência de seu embarque.*

Em um ambiente ubíquo, como o do projeto Aura da Carnegie Mellon University [15], seria possível verificar a impossibilidade de transferência completa do documento de Luciana antes da partida de seu vôo e, através de observações na progamação de decolagens do aeroporto e avaliações climáticas e de tráfego aéreo para previsão de atrasos, orientar a usuária a dirigir-se ao portão mais próximo cujo fluxo na rede não impeça a total conclusão de suas atividades. Esta pró-atividade do sistema agrega conforto e segurança à usuária, na medida em que garante o cumprimento de seus compromissos e proporciona-lhe maior comodidade.

2.2 Propriedades

A proposta dos sistemas ubíquos surge em meio à intensa pulverização de dispositivos móveis em contextos do mundo real. Para efetivamente caracterizar o ideal de Weiser e distingui-lo de outras novidades computacionais, pesquisadores de todo o globo registram, em suas publicações acadêmicas, características essenciais ao verdadeiro cenário ubíquo.

2.2.1 Integração física

Espaços inteligentes, os *smart spaces*, reduto precursor da computação ubíqua, são naturalmente definidos pela colaboração intensa entre elementos computacionais e componentes do mundo físico. Em [16], Kindberg e Fox definem esta integração como ponto irrefutável à caracterização da *ubicomp*.

Para exemplificar, citam o projeto MediaCups [17], que demonstra como um objeto físico do mundo real pode infiltrar-se em um cenário lógico computacional. No projeto, uma xícara de café que, como componente do mundo físico, assume seu papel natural de recipiente de café mas que, como integrante de um universo lógico, também é dotada de sensores e recursos de processamento e rede, que a permitem comunicar o estado de seu usuário aos demais elementos, físicos ou lógicos, do espaço inteligente.

2.2.2 Invisibilidade

Quanto mais presente em nossas vidas uma tecnologia estiver, menos perceptível ela deve ser [13]. Não é por acaso que se define a computação ubíqua também como computação invisível. Nesta ótica, o computador torna-se pedra fundamental das atividades cotidianas mas dilui-se no mundo físico tornando-se tão presente e imperceptível quanto o ar que se respira.

Em termos gerais, extingue-se a integração explícita entre o homem e o computador e deixa-se que ambas entidades convivam em perfeita simbiose. Quanto mais alheio o homem estiver das máquinas que o rodeiam, melhor. Assim como os símbolos romanos da escrita nos passam despercebidos no dia-a-dia, a informática também deve ser consumida inadvertidamente.

Em 1995, Mark Weiser e John Brown defenderam a computação ubíqua com base nos conceitos de periferia e centro de nossas atenções [18]. Por eles, para que a tecnologia efetivamente se infiltre em nossas vidas, ela não deve exigir mais que nossa atenção periférica. Assim, qualquer *middleware* que trabalhe na integração de elementos de um universo ubíquo deve primar pela discrição em seu funcionamento.

2.2.3 Pró-atividade

Na computação ubíqua, define-se por pró-atividade a capacidade do sistema em antecipar a intenção do usuário, como exemplificado na situação hipotética de Luciana, na seção 2.1.

2.2.4 Sensibilidade ao contexto

Para que um sistema pervasivo, base dos ambientes ubíquos, tenha o mínimo de intrusão no mundo real, sua infra-estrutura de software deve ser sensível ao contexto [19], ou seja, possuir uma base extensa de informações a respeito das pessoas e do ambiente que as abriga e, através destes dados, adaptar seu comportamento da melhor forma possível para completa satisfação das espectativas de seus usuários.

Em um modelo que busca a onipresença computacional, o sucesso da pró-atividade do sistema é alavancado pela completude das informações de que dispõe. No exemplo de Luciana, seção 2.1, observamos uma suave integração do conhecimento de sistemas de diferentes níveis computacionais, como softwares de base para a identificação do congestionamento da rede *wireless*, sensores físicos para captação das condições meteorológicas e aplicativos corporativos para a avaliação do cronograma de vôos. Toda esta cooperação, quando implementada de forma ágil e transparente, traz ao usuário final a antecipação e automação da tomada de decisões, que lhe reservam tempo hábil para dedicação prioritária à sua atividade fim.

A riqueza de um contexto da computação ubíqua pode ser constituída de um conjunto de atributos físicos (e.g., localização de pessoas e dispositivos no espaço), fisiológicos (e.g., temperatura corporal e batimentos cardíacos), psicológicos (e.g.,

alegria, ansiedade, angústia, irritação), histórico de atividades, padrão de comportamento, entre outros [14]. Com esta bagagem de informações, assim como uma secretária executiva deve ser capaz de antecipar os desejos de seu chefe e, diante de um cenário de irritação, incomodá-lo apenas em situações de emergência, a computação ubíqua também deve buscar a coleta de dados para evitar ao máximo a necessidade de chamar a atenção de de seus usuários.

2.2.5 Interoperabilidade espontânea

Ainda segundo Kindberg e Fox [16], um sistema ubíquo é simploriamente fragmentado em componentes de infra-estrutura, de natureza fixa em relação ao *smart space*, e componentes espontâneos, entendidos como unidades de software para abstração de serviços, recursos ou dispositivos em constante deslocamento entre ambientes distintos.

Dentro desta definição, é apresentada a interoperabilidade espontânea como característica desejável a implementações da computação ubíqua, observada pela possibilidade de integração dinâmica entre componentes móveis e de infra-estrutura do sistema.

Seja um ambiente ubíquo implementado em uma sala de reuniões de uma empresa. A interoperabilidade espontânea pode ser observada se, por exemplo, um cliente ou um fornecedor da organização é capaz de transferir a apresentação armazenada em seu PDA ao projetor da sala, sem qualquer intervenção manual para configurar o dispositivo.

Os autores ressalvam ainda a preferência do termo *espontâneo* ao termo *ad hoc*. Por eles, *ad hoc* tradicionalmente refere-se a arquiteturas de redes, independentes de infra-estrutura fixa, enquanto a idéia da espontaneidade agrega mais amplitude ao termo, numa noção não exclusivamente infra-estrutural, como também de serviços que dinamicamente se encontram, se conhecem e se interagem para a troca de informações.

2.2.6 Interfaces naturais

Em [9], Mark Weiser já discursava sobre a idéia de que "o mundo não é um *desktop*". A artificialidade envolvida no modelo de janelas da computação moderna infringem o princípio básico de invisibilidade da computação ubíqua. Em *smart-spaces*, é preciso que se busque técnicas para que os recursos normalmente utilizados no dia-a-dia de uma sociedade, como gestos, voz e mesmo olhares, permaneçam como meio de comunicação entre homem e máquina. Se a proposta é integrar sutilmente elementos digitais ao mundo real sem influenciar na normali-

dade de seu funcionamento, então é mister a manutenção de interfaces naturais [20] de interatividade entre o homem e o mundo que o rodeia. A inclusão de ferramentas doutrora desnecessárias à condução do cotidiano humano, como interfaces gráficas de interação (GUI), através de periféricos de entrada de dados (e.g. mouse e teclado), não são justificadas sob a premissas essenciais da *ubicomp*.

A computação ubíqua é então um novo paradigma computacional fundado sobre a ampla integração da informática com objetos do mundo físico e sobre uma intensa mobilidade nos espaços que a contemplam. As propriedades listadas neste capítulo não necessariamente refletem a totalidade das virtudes requeridas à *ubicomp*, mas representam seus principais atributos. No próximo capítulo, serão descritos alguns projetos de maior relevância deste universo de pesquisa que demonstram como a comunidade tem caminhado rumo a esta nova era.

Capítulo 3

Projetos de Computação Ubíqua

> "Like the frontier of the American West in the early 19th century, pervasive computing offers new beginnings for the adventurous and the restless - a rich open space where the rules have yet to be written and the borders yet to be drawn."
>
> M. Satyanarayanan, 2001

Em face aos desafios sócio-culturais e tecnológicos impostos pelas novidades da computação invisível, as comunidades de pesquisas se agitam em busca de padrões e soluções capazes de promover a computação a um novo estágio de integração com o mundo real. A fim de amenizarem as consequências dos desafios apresentados no capítulo 2, algumas iniciativas público e privadas tornam-se referências internacionais e fixam-se na vanguarda da computação ubíqua, colaborando na definição dos rumos evolutivos da tecnologia.

Desde 1988, a computação ubíqua vem sendo fonte de especulação por promover transformações tão profundas na realidade técnica e social humana. Por sua imaturidade, quando comparada a outras linhas de pesquisas científicas, inúmeros problemas ainda econtram-se em aberto. Como ainda não existe, no universo da computação ubíqua, organismos reguladores, os trabalhos giram em diferentes órbitas na busca de padrões *de facto* para a comunidade. Grandes centros de pesquisas já iniciaram a corrida contra o tempo a fim de fixarem seus ideais como definições internacionais. A exemplo, cita-se os projetos do presente capítulo que, com focos de propostas tão distintos, caracterizam a amplitude dos esforços conduzidos na área.

Quando vista sob a luz da computação distribuída, da computação paralela, ou mesmo da computação móvel, a computação ubíqua demonstra sua juventude em termos da disponibilidade de literatura específica, da quantidade de periódicos especializados, do número de universidades envolvidas e da quantidade de padrões já consolidados. Pela lista dos projetos mais relevantes, que compõem

seu estado da arte, é possível observar ainda a ausência de grandes desafios que direcionem a coletividade. Aparentemente, a meta primeira é a concretização do conceito da computação ubíqua pois, o que se observa, outrossim, é a busca do desenvolvimento de cenários que tangibilizem sua realidade, viabilidade e oportunidades. E, dentre as várias iniciativas, cada grupo prioriza características que acreditam ser os pontos mais relevantes desses cenários. Por exemplo, no projeto Aura, toda argumentação gira em torno da manutenção da lista de tarefas a serem executadas diariamente pelos usuários da *ubicomp*. Já para o projeto Gaia, as atenções firmam-se sobre o desenvolvimento de uma infra-estrutura base para a contrução de aplicativos. O projeto EasyLiving, por sua vez, prima pela integração dinâmica de dispositivos para o enriquecimento da experiência do usuário e, o projeto Gator Tech, propõe ambientes que reajam à alteração do comportamento de seus integrantes. Importante ressaltar, entretanto, que a ausência de foco de um ou outro projeto em determinado atributo não configura sua deficiência naquele item. Quer dizer, por exemplo, que o fato de o projeto Aura não focar na viabilização de uma biblioteca de desenvolvimento não necessariamente implica que ele não possua esta infra-estrutura de programação, mas sim que este não é o cerne de sua pesquisa.

Abaixo estão listados os principais projetos de *ubicomp* citados na literatura científica.

3.1 Aura

O projeto Aura [2, 21, 22, 15, 23], desenvolvido na *Carnegie Mellon University*, busca primordialmente a otimização do uso da atenção humana, por eles categorizada como o recurso mais escasso em sistemas computacionais. Para isso, definem uma infra-estrutura para a acomodação de sistemas ubíquos, com premissas básicas em pró-atividade e auto-ajuste do sistema às necessidades do usuário.

Pela visão do projeto, as pessoas possuem diariamente um conjunto de tarefas a cumprir e deslocam-se constantemente de um ponto a outro, conforme a atividade em execução. Neste cenário, o projeto Aura explora um modelo para a representação formal deste conjunto de tarefas, capturando as necessidades do usuário em termos dos serviços requeridos do *smart-space*, suas interconexões, preferências pessoais e níveis de qualidade de serviço. A medida que o usuário migra de um espaço para outro, o ambiente verifica a atividade em execução e busca no meio a combinação de recursos necessária à execução da tarefa em curso.

A arquitetura do middleware é apresentada na figura 3.1, onde, logicamente, destacam-se 4 módulos principais:

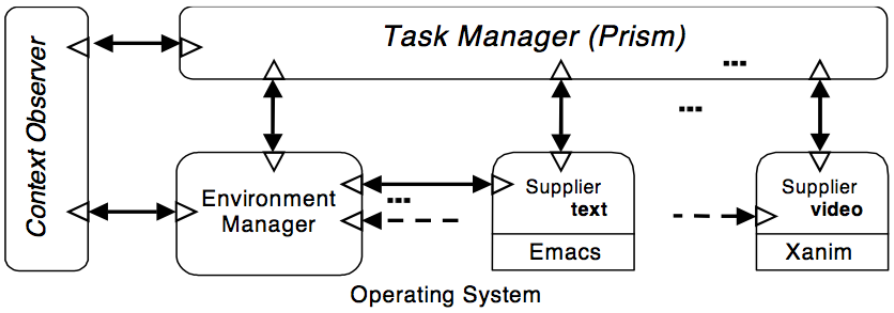

Figura 3.1: A arquitetura do projeto Aura [2]

- *Task Manager* - Reponsável por monitorar a intenção do usuário com base em suas preferências e em sua previsão de tarefas a serem executadas.
- *Environment Manager* - Provê a melhor combinação de serviços do ambiente para realização das tarefas identificadas pela camada *Task Manager*.
- *Environment* - Gerencia as aplicações ou dispositivos (*suppliers*) que implementam os serviços descritos na camada *Environment Manager*.
- *Context Observer* - Mantém a base de informações sobre o estado momentâneo do usuário.

3.2 Gaia

Definido formalmente como um *middleware operating system* baseado em componentes, o projeto Gaia [3, 24] da *University of Illinois at Urbana-Champaign*, ou simplesmente GaiaOS, define uma infra-estrura distribuída de software para a construção e disponibilização de aplicações ubíquas.

Através de um mecanismo para a coordenação e homogenização do acesso aos recursos do ambiente, o projeto converte o espaço físico e todos os elementos de hardware que nele residem em um ambiente computacional programável chamado de *active space*. Semelhante a um sistema operacional, que abstrai o acesso aos recursos físicos de um computador, o projeto Gaia pode ser compreendido como um sistema operacional para ambientes ubíquos, responsável pela abstração do acesso aos recursos físicos do *smart-space*. Vários dos conceitos implementados pelos sistemas operacionais modernos, como eventos, sinais, sistemas de arquivos, processos etc, são implementados pelo projeto no contexto da computação ubíqua.

As atividades do projeto são distribuídas entre a construção do *middleware GaiaOS*, do framework de desenvolvimento de aplicações *Gaia Application Model* e a linguagem de script para integração destes elementos *LuaOrb*.

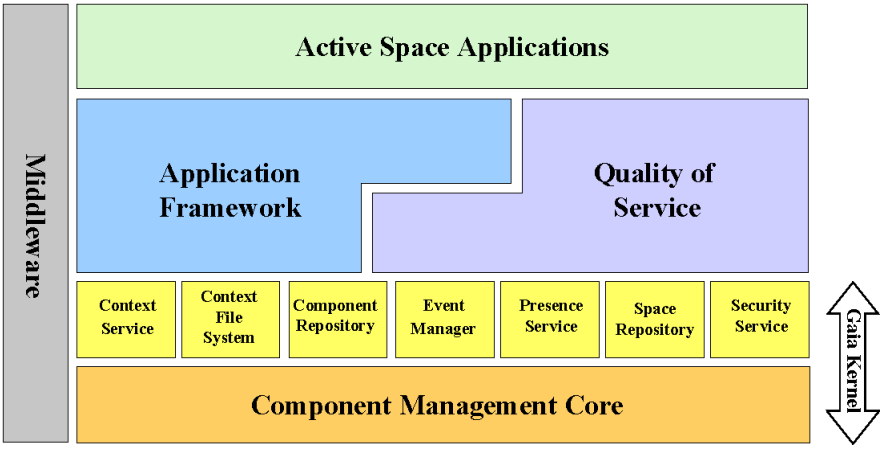

Figura 3.2: A arquitetura do projeto Gaia [3]

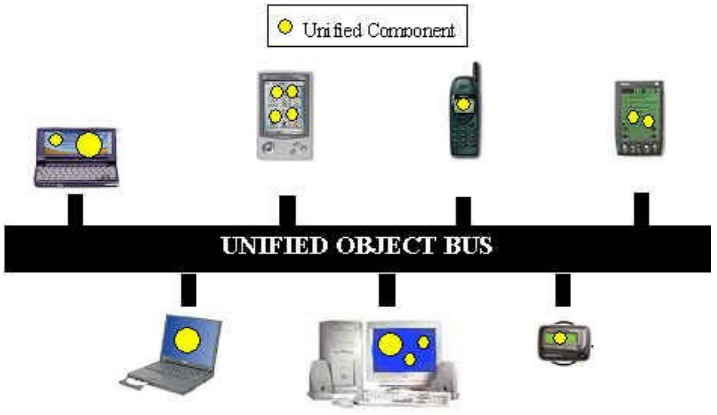

Figura 3.3: *Unified Object Bus* do projeto Gaia [3]

A arquitetura *GaiaOS*, apresentada na figura 3.2, é fundamentada em dois elementos principais: um barramento comunicação entre objetos (*Unified Object Bus*) e o núcleo do sistema (*GaiaOS Kernel*).

O *Unified Object Bus*, figura 3.3, estabelece um mecanismo comum para acesso a todos os tipos de objetos do *active space*. Independente do modelo de componentes utilizado para o desenvolvimento de aplicações dentro do ambiente Gaia (CORBA [25], DCOM [26], EJB [27]), este barramento pavimenta a via para a livre comunicação entre todas as entidades de software do meio. Em suma, é por meio do *Unified Object Bus* que é possível a integração de dispositivos de diferentes fabricantes, com diferentes sistemas operacionais e diferentes implementações de tecnologias de *middleware*.

No nível do kernel, *GaiaOS Kernel*, são definidos todos os serviços essenciais para inicialização básica do sistema, como gerenciador de eventos; localizador de serviços e indivíduos; base de registro de dispositivos, serviços, indivíduos e aplicações ativos no sistema; sistema de arquivos; e serviços de autentitcação e autorização de usuários.

O framework de desenvolvimento de aplicações, *Gaia Application Model*, também parte do projeto, baseia-se em uma adaptação do padrão MVC (*Model View Controller*) [28] para a realidade da computação pervasiva, chamada de MPACC (*Model Presentation Adapter Controller Coordinator*) [14]. Na visão do desenvolvedor, o modelo de programação especifica como as aplicações deverão ser projetadas, desenvolvidas e interligadas. Para os usuários do sistema, o framework facilita a utilização e customização dos aplicativos existentes.

Para coordenar as aplicações e componentes do sistema operacional ubíquo, o projeto *Gaia* faz uso de um ambiente de programação dinâmico, baseado em *script*, chamado LuaOrb. Semelhante à linguagem BASH [29], a linguagem Lua, base do LuaOrb, disponibiliza recursos para a configuração de serviços, coordenação do processo de inicialização, reconfiguração do sistema, em caso de mudanças no ambiente, prototipação rápida de novas aplicações, extensão de componentes e realização de testes em geral.

Vários vídeos que demonstram o funcionamento do ambiente *Gaia* estão disponíveis na página do projeto na Internet. Segundo [24], apesar da imensa quantidade de problemas ainda não solucionados nesta área de pesquisa, os resultados já obtidos demonstram o quão benéfica a construção de uma infra-estrutura comum de desenvolvimento pode ser para a implementação de soluções de computação ubíqua.

3.3 Gator Tech

The Gator Tech Smart House é um projeto de uma casa inteligente conduzido pelo Laboratório de Computação Móvel e Pervasiva em colaboração com o *College of Public Health and Health Professions* da Universidade da Flórida [4]. Seu objetivo central é a criação de ambientes assistidos computacionalmente, capazes de reação com base no monitoramento de seu estado e de seus habitantes.

Tecnicamente, uma das motivações originárias do projeto foi o desenvolvimento de um ambiente pervasivo dinâmico, capaz de evoluir e se adaptar a tecnologias emergentes, ao contrário, segundo os autores de [4], das primeiras gerações de sistemas ubíquos, estáticos no tempo e extremamente frágeis à adoção de novos

padrões e novos dispositivos.

O resultado desta motivação foi o desenvolvimento de um *middleware* programável, no qual a arquitetura ubíqua é composta tanto por módulos executáveis, quanto por bibliotecas de programação, para a fácil evolução do sistema.

A arquitetura do *middleware* do projeto é apresentada na figura 3.4. Ao todo, são seis módulos de software distribuídos em camadas para viabilização do sistema de execução do *smart-space* e do modelo de programação orientado a serviços, através dos quais serão feitos o registro e a descoberta de serviços, o acesso aos recursos físicos do meio e garantida a consistência do estado de execução do sistema, dentre outros.

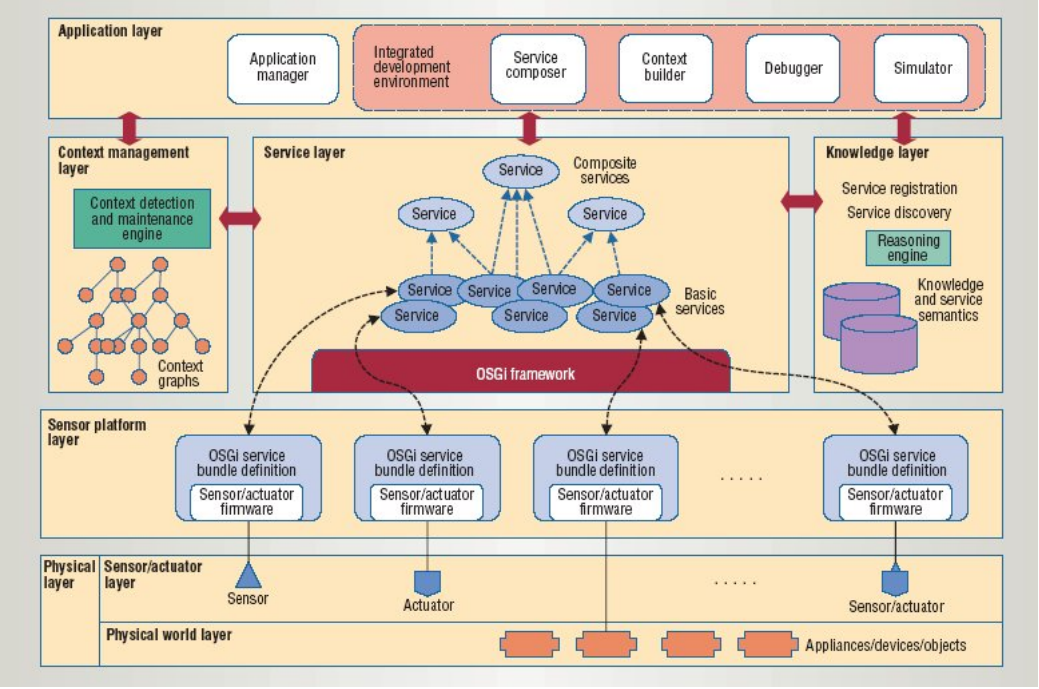

Figura 3.4: A arquitetura do projeto Gator Tech [4]

Cada camada da arquitetura é descrita a seguir:

- *Physical layer* - Consiste nos elementos de hardware e demais objetos que compõem o *smart-space*, como lâmpada, TV, campainha, detector de fumaça, aquecedor, móveis, eletrodomésticos, entre outros.

- *Sensor platform layer* - Esta camada encapsula todos os detalhes de baixo nível da *Physical layer*, estabelecendo a comunicação desta com as camadas superiores da arquitetura, possibilitando o desenvolvimento de serviços independentes do hardware que o implementa. Todo dispositivo ligado ao

smart-space é exposto no *middleware* como uma classe Java. Para cada dispositivo, corresponde um *sensor platform*, construído sobre uma memória programável (EEPROM), dentro da qual é mantido o *driver* do dispositivo, *bytecode* Java que contém informações estáticas sobre o dispositivo e os serviços que implementa. Ao ser acionado, a *sensor platform* registra o *driver* do dispositivo no *middleware* do *smart-space*, que é então disponibilizado para o desenvolvimento de novos serviços.

- *Service layer* - Nesta camada são mantidos os serviços básicos publicados pela *sensor platform*, bem como outros serviços formados pela composição dos primeiros. Internamente, sua arquitetura é baseada numa implementação do framework OSGi [30], que define um ambiente computacional, orientado a componentes, para a disponibilização de serviços de rede.

- *Konwledge layer* - Representa o cérebro do *middleware*. Nele é mantido o registro de todos os elementos do *smart-space*, organizado como uma ontologia, por meio da qual garante-se a descoberta de serviços baseada em inferências, através de um sub-módulo chamado de *reasioning engine*.

- *Context management layer* - Responsável pelo gerenciamento de contextos do sistema, que podem ser criados a qualquer tempo, com base nos interesses dos desenvolvedores. Através de um *context engine*, o estado do *smart-space* é constatemente monitorado, a fim de que o sistema não entre em cenários de inconsistência, como poderia ocorrer caso um aquecedor e um ar condicionado fossem acionados simultaneamente.

- *Application layer* - Gerenciador de aplicações do sistema. Permite a ativação e desativação de serviços, e fornece um conjunto de ferramentas gráficas para o desenvolvimento interativo de novas funcionalidades, definição de novos contextos, depuração e simulação de situações.

3.4 ISAM

Em nível nacional, o projeto ISAM (Infra-Estrutura de Suporte às Aplicações Móveis) [5] é um dos de maior destaque. Sua proposta é o fornecimento de uma arquiteutra que simplifique a tarefa de implementação de aplicativos móveis com comportamento adaptativo.

Segundo seus autores, na computação móvel, é intensa a flutuação da disponibilidade dos recursos computacionais do ambiente e, por esta razão, é complexa a

tarefa de desenvolvimento de aplicações para esses cenários. Dessa forma, defendem que as aplicações construídas para ambientes de computação móvel devem ser auto-ajustáveis às constantes oscilações de recursos do meio e, por esta razão, descrevem a capacidade de adapção das aplicações como uma propriedade fundamental para a computação móvel.

Apesar de não mencionar o termo *computação ubíqua*, mas apenas *computação pervasiva*, o projeto ainda assim foi considerado para o escopo deste trabalho pois percebe-se que o termo não foi utilizado apenas no contexto da alta integração do sistema com o meio físico [1], mas sim num cenário de mobilidade física e lógica onde a conexão de rede é sempre mantida.

A justificativa para criação de uma arquitetura específica para ambientes móveis distribuídos vem da constatação de que as tradicionais propostas da computação distribuída não satisfazem às particularidades da computação móvel. Assim, para melhor usufruto da mobilidade, propõem uma arquitetura que possibilita a adaptação do sistema e de suas aplicações às inconstâncias da disponibilidade de recursos dos ambientes móveis, para manutenção da qualidade dos serviços providos aos usuários presentes.

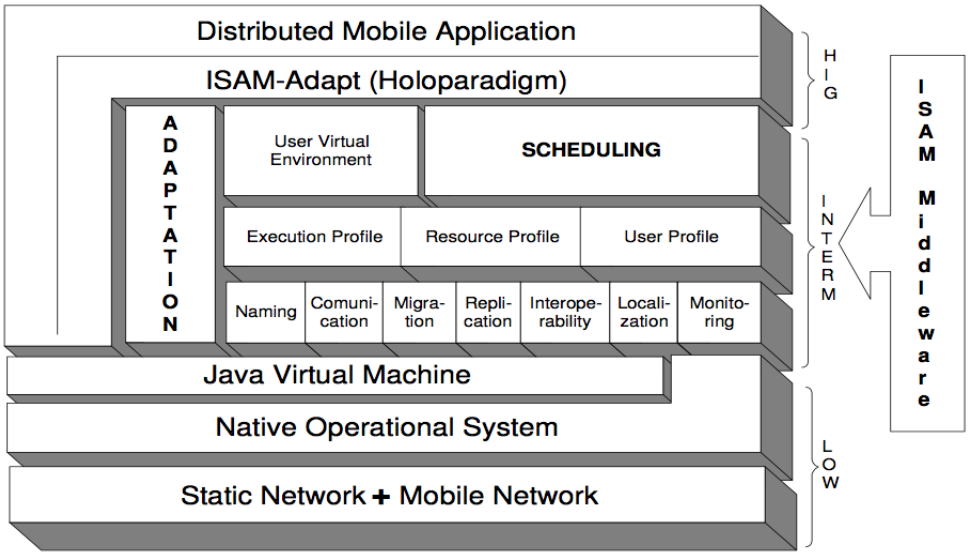

Figura 3.5: Arquitetura do projeto ISAM [5]

Nesta arquitetura modular, apresentada na figura 3.5, três níveis lógicos de componentes são propostos. No nível mais baixo, encontram-se os sistemas existentes, como as tecnologias de comunicação móvel, sistemas operacionais e a máquina virtual Java, sobre os quais é desenvolvido o restante do projeto. No extremo oposto, no nível mais alto da arquitetura, estão as aplicações, desenvolvidas sobre uma abstração capaz de expressar a mobilidade e a adaptabilidade

de forma natural e explícita no código, chamada de *HoloParadigm*. Entre essas duas camadas, existe um nível intermediário, que constitui o *middleware* da proposta, chamado de EXEHDA (*Execution Environment for Highly Distributed Applications*), no qual se mantém todos os módulos funcionais do projeto, abaixo resumidamente descritos:

- *Monitoring* – Captura as informações de contexto de usuários e aplicações.
- *Localization e Naming* – Provêem suporte ao deslocamento físico dos usuários entre células da arquitetura móvel.
- *Replication* – Possibilita maior disponibilidade e performance do acesso aos dados.
- *Migration* – Promove o deslocamento físico de componentes.
- *User Profile, Resource Profile e Execution Profile* – Fornecem, com base nos dados capturados dos módulos inferiores, as informações necessárias à tomada de decisão sobre a adaptação.
- *User Virtual Environment* – Composto por elementos que integram a interface móvel do usuário, apresentando-lhe informações ajustadas às suas necessidades e contexto.
- *Scheduling* – Integra todo o sistema às aplicações para a adaptação colaborativa e multi-nível.

Para responderem às questões sobre quem executa a adaptação e como a adaptação é realizada, Yamin et. al. [5] sugerem uma taxonomia (figura 3.6) para aplicações móveis com comportamento adaptativo. Por esta, a responsabilidade da adaptação é compartilhada entre o sistema e as aplicações. Os elementos motivadores à adaptação são então classificados em recursos (rede, arquivos, dispositivos, aplicativos etc), contexto (toda e qualquer informação relevante à aplicação para definir seu comportamento) e localização (dados sobre o posicionamento da aplicação e seus usuários no espaço). Deste ponto, definem três conjuntos macros de aplicações, com base em sua sensibilidade à adaptação:

- *resource-aware applications* – Aplicações que se adequam à disponibilidade de recursos do ambiente. Sua adaptação busca transparecer aos usuários eventuais indisponibilidades que ocorram. A maioria desses casos de adaptação podem ser executados pelo próprio sistema.

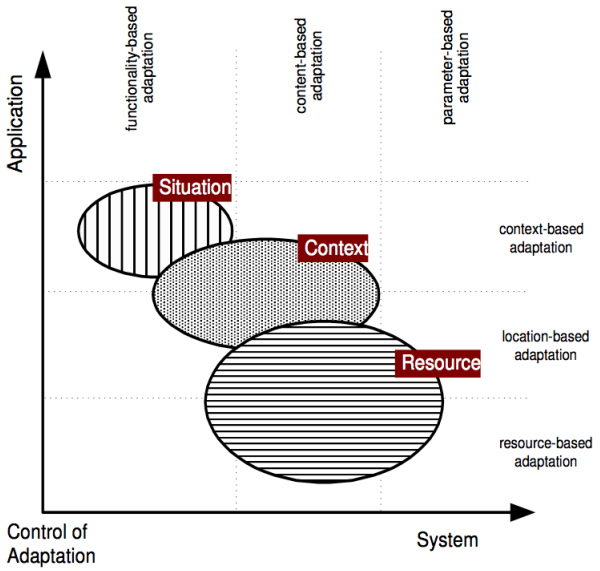

Figura 3.6: Taxonomia para aplicações móveis com comportamento adaptativo [5]

- *context-aware applications* – Aplicações que ajustam seu comportamento com base em variações internas e externas, observadas através sensores e monitores.

- *situation-aware applications* – Aplicações que se adaptam por consequência de coleta de dados feita de outras aplicações, do contexto de sua utilização e das preferências do usuário.

Assim, o projeto apresenta sua abordagem uniforme, colaborativa e multi-nível de adaptação, com foco independente de qualquer domínio de problema e baseada em uma taxonomia original e inovadora.

3.5 EasyLiving

Este projeto define um conjunto de tecnologias destinadas à integração dinâmica de dispositivos para o enriquecimento da experiência do usuário na utilização do espaço [31].

Para exemplificarem seu funcionamento, Brumitt et al. descrevem um cenário onde Tom entra no ambiente do EasyLiving e, pela interface de um desktop, seleciona uma lista de músicas a serem reproduzidas. Em seguida, ao se dirigir ao sofá e lá se sentar, a lista de reprodução é automaticamente projetada na parede à sua frente, para que seu controle seja mantido por Tom, demonstrando, assim, a sintonia entre o desktop e os demais recursos presentes no ambiente.

Construído nos laboratórios de pesquisa da Microsoft, o projeto sedimenta-se no conceito de ambientes inteligentes, que é definido como um espaço que contém dispositivos fixos e móveis que se integram para o fornecimento de informação aos seus usuários. Neste contexto, o objetivo do projeto EasyLiving é permitir, através de uma arquitetura pré-estabelecida, que cada vez mais dispositivos possam se integrar a este ecosistema de elementos que relaciona o usuário com o ambiente inteligente que o abriga.

A estrutura do projeto é baseada em quatro pilares: *middleware*, modelagem geométrica, sensoreamento e descrição de serviços, conforme descritos a seguir.

Várias tecnologias de *middlewares* estão hoje disponíveis no mercado, como CORBA, DCOM e Java (RMI, EJB, Jini etc). Entretanto, por terem sido concebidas para a computação distribuída tradicional, essas tecnologias falham na completa satisfação das necessidades das computação ubíqua, como o tratamento de problemas gerados pela alta mobilidade e heterogeneidade de dispositivos. Então, em reação a esta deficiência, a Microsoft desenvolveu uma infra-estrutura de *middleware* chamada *InConcert*, agregada ao projeto EasyLiving, com suporte a mecanismos para as especificidades da *ubicomp*. Em especial, a tecnologia *InConcert* provê recursos de mensageria assíncrona, endereçamento de objetos independente de dispositivos e utilização do formato XML para a troca de mensagens entre os componentes da arquitetura.

No modelo computacional baseado em desktops, assume-se que todos os dispositivos de I/O estão fisicamente interconectados. Em um ambiente distribuído de computação ubíqua, esta premissa perde sua validade, haja vista a dinâmica de mobilidade do ambiente. Para estabelecer então o modelo de inteconectividade entre dispositivos destes cenários ubíquos, o projeto EasyLiving adotou o estudo da geometria do ambiente como ferramenta principal.

Na modelagem geométrica reside a consolidação das informações sobre o relacionamento entre pessoas, dispositivos e o ambiente. O modelo geométrico do espaço, provido pelo projeto EasyLiving, permite a micro localização de entidades presentes no ambiente para a descrição de seu posicionamento relativo a outras entidades e a identificação das entidades mais adequadas para satisfação de alguma demanda. Por exemplo, no cenário descrito acima, foi graças à modelagem geométrica do ambiente que, ciente da localização do usuário e dos dispositivos presentes na sala, o sistema EasyLiving pôde identificar que a parede em frente ao sofá era, naquele momento, o melhor lugar para projeção do controle da lista de reprodução musical de Tom.

Para a coleta das informações necessárias à manutenção deste modelo geomé-

trico do ambiente, o sistema conta com uma camada de sensoreamento distribuída por todo o espaço, da qual fazem parte câmeras de vídeo e dispositivos móveis com suporte à rádio frequência. Através das câmeras de vídeo, o sistema identifica e rastreia todas as entidades móveis presentes no ambiente. Pela tecnologia de rádio frequência, com base na intensidade do sinal recebido de bases fixas, identifica-se a localização de dispositivos difíceis de serem rastreados visualmente pelas câmeras.

No EasyLiving, todas as funcionalidades do ambiente são encapsuladas em serviços lógicos cujo projeto desacopla o controle físico do dipositivo, a lógica do serviço e sua interface de interação. Cada serviço possui um conjunto de comandos que são expostos no ambiente para sua integração com outros serviços do ambiente. Para a descrição do serviço e suas funcionalidades, estruturas em XML são utilizadas, o que possibilita a realização de consultas de serviços com base em seus atributos e a identificação do possíveis valores dos parâmetros para sua utilização.

3.6 Outros trabalhos

A seleção do grupo de projetos analisados neste capítulo não se baseou em critérios objetivos, mas apenas em observações quantitativas feitas sobre as publicações mais evidentes do segmento. Acredita-se, entretanto, que a amostragem representa as principais propostas de desenvolvimento da computação ubíqua. Todavia, é importante destacar outros projetos e grupos de pesquisa, que não fizeram parte desta análise, mas que também possuem relevância na área, como o projeto Ninja [32] e Endeavour [33] da Berkeley University, PICO [34] da *University of Texas at Arligton*, Oxygen [35] e Things that Think [36] do MIT, Portolano [37] e one.world [38] da *Washington University*, CoolTown [39] da HP, iRoom [40] de *Stanford University*, Future Computing Environments [41] do Georgia Institute of Technology, Next Wave [42], Equator [43], 2K [44], o Integrated Project on Pervasive Gaming [45], Uk-UbiNet [46] e Ubiquitous Computing Evaluating Consortium [47].

Capítulo 4

Adaptabilidade de Serviços

"Getting the computer out of the way is not easy."
Mark Weiser, 1993

Este capítulo define os conceitos chaves para compreensão do restante do trabalho e analisa como os projetos do capítulo anterior os contemplam.

4.1 Conceitos

Devido ao fluxo contínuo de dispositivos e pessoas no perímetro de um *smart-space*, seu sistema precisa estar sempre se adaptando às mudanças de estado do ambiente. Para o bom atendimento às necessidades de seus usuários, é importante que os serviços do meio ajustem suas configurações conforme as variações do conjunto de entidades que compõem o espaço.

Fouial, Fadel e Demeure [11], ao proporem uma plataforma para o provimento de serviços reconfiguráveis, definem três conceitos de suma importância para a formação do vocabulário aqui utilizado:

Adaptação de serviço - Consiste na adequação do serviço às mudanças de seu ambiente de execução. Isto é, seja um ambiente em um estado E no qual é executado um serviço S. Caso algum fator altere o estado do ambiente para E', chama-se de *adaptação* ao processo de ajuste do serviço a uma nova configuração S': E x E' x S -> S'

Adaptabilidade de serviços - Facilidade do serviço para se adaptar às mudanças do ambiente de execução. Ou seja, se o sistema possuir uma arquitetura que favoreça e simplifique a *adaptação de serviços*, ele estará colaborando com a *adaptabilidade de serviços* daquele ambiente.

Serviço adaptativo - Característica dos serviços que se adaptam automaticamente. A *adaptação do serviço* pode ser iniciada manualmente pelo usuário ou automaticamente pelo próprio sistema, com base em informações lidas do contexto.

É fácil observar que a terminologia acima em nada depende da computação ubíqua. Tratam-se de conceitos genéricos, aplicáveis a quaisquer áreas de pesquisa da informática.

No geral, a grande meta da comunidade de pesquisa é o desenvolvimento de serviços adaptativos. Entretanto, para que exista a adaptação automática de serviços, é preciso que antes haja a implementação manual desta adaptação, o que só é possível através da incorporação de recursos na arquitetura que viabilizem sua fácil reconfiguração. A figura 4.1 ilustra o encadeamento lógico de dependências entre os três conceitos vistos acima.

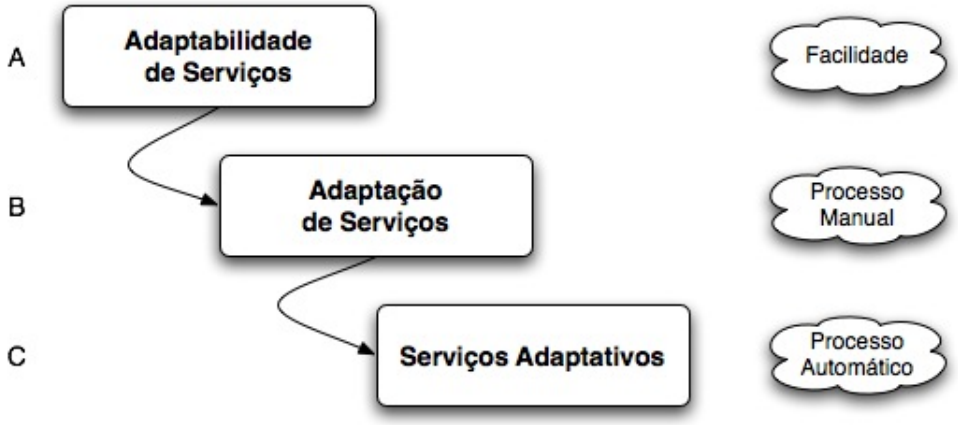

Figura 4.1: Relação entre Conceitos

A figura demonstra que a facilidade da arquitetura em se reajustar às mudanças do ambiente (A) viabiliza a adaptação manual de serviços (B), que é, por sua vez, base para a automatização do processo (C). Ou seja, não é possível automatizar algo que sequer é feito manualmente. E também, não se implementa a adapção manual de serviços caso a arquitetura do sistema seja monolítica e de pouca flexibilidade.

A adaptabilidade de serviços então, apesar de explicitamente pouco explorada, é a base para todo este processo de seleção automática de serviços.

Quando se fala da adaptabilidade, fala-se da maleabilidade da infra-estrutura do sistema para a reassociação entre serviços e clientes, ou seja, a facilidade de clientes do ambiente em efetivarem a troca de provedores dos serviços que utilizam.

Havendo esta facilidade, é possível então implementar o processo de reestabelecimento dos laços de dependências entre as entidades de software do ambiente e, daí para sua automatização, é uma questão de implementar os melhores mecanismos para esta tomada de decisão.

4.2 Adaptação de Serviços

Os trabalhos relacionados à adaptação automática de serviços, não necessariamente relacionados à computação ubíqua, pautam-se geralmente em técnicas que se utilizam de recursos matemáticos, estatísticos e de inteligência artificial, que viabilizam a seleção dinâmica dos melhores provedores de um certo serviço.

A exemplo, na tecnologia de Markus Huebscher e Julie McCann [48], clientes podem quantificar a confiabilidade de um provedor de serviços e, dinamicamente, decidir ou não por sua escolha.

Nesta arquitetura, para a utilização de um serviço, é adotada uma função de utilidade `u(QoI, d)` que calcula o melhor provedor do serviço requerido. Nesta função, `QoI`, *Quality of Information*, é um índice que indica a qualidade da informação desejada pelo cliente e `d` é um valor por eles chamado de *dependability*, que aponta a satisfação do cliente com a informação provida pelos *service providers* que está utilizando.

Para demonstrar o funcionamento deste mecanismo, Huebscher baseou-se em serviços de metereologia. Dois provedores de dados metereológicos são disponibilizados e o sistema decide utilizar um ou outro com base nos cálculos de sua função de utilidade, que tenta mensurar a precisão da estimativa do tempo.

4.3 Adaptabilidade de Serviços

No geral, uma característica observada nos trabalhos sobre a adaptação automática de serviços é o grande foco dado aos mecanismos de seleção dos melhores provedores de serviços e a pouca atenção dispensada aos meios que viabilizam essa possibilidade. Esses meios, que juntos conferem a adaptabilidade de serviços ao ambiente, são vez ou outra citados em artigos, mas raramente como instrumento fim da pesquisa.

Esta seção busca apresentar alguns trabalhos nesta linha de pesquisa e resgatar o tópico adaptabilidade de serviços de dentro dos projetos de maior relevância no mundo da computação ubíqua, vistos no capítulo 3, analisando o que existe nas arquiteturas propostas que contribui para viabilizar a adaptação de serviços, conforme definida na seção anterior.

4.3.1 Hubscher e McCann

Em outro artigo [49] e sem perder o foco na seleção automática de serviços, Hubscher e McCann propõem uma arquitetura de *middleware* para a identificação e seleção dos melhores provedores de informações de contexto, baseada em quatro camadas, assim descritas:

- Sensor – Nível mais baixo da arquitetura, no qual encontram-se dispositivos para o sensoreamento do meio e captação de dados de contexto.
- Provedor de contexto (CP) – Compila os dados fornecidos pela camada de sensores, interpretando-os e disponibilizando-os, na forma de serviços, às aplicações interessadas.
- Serviço de contexto (CS) – Camada para o desacoplamento entre provedores de contexto e aplicações. Através deste nível de indireção, é possível que o provedor de contexto de uma aplicação seja transparentemente substituído. Existe apenas um CS para um conjunto de CPs que fornecem o mesmo tipo de informação.
- Aplicação (APP)– Software utilizado pelo sistema ou pelo usuário.

Acompanham essas camadas dois outros componentes da arquietura: um módulo de descoberta de serviços (SD) e outro resposável pelo mecanismo de adaptação de serviços (AE).

Ao entrar no *smart-space*, os CPs são automaticamente registrados no SD. Equanto estiverem ativos, cada CP deve enviar-lhe pulsos periódicos, notificando-o de sua existência.

Quando uma APP entra na rede do *smart-space*, ela envia ao SD uma listagem com os tipos de contexto de que necessita (temperatura, localização etc). Para cada contexto enviado, o SD busca na rede um conjunto de CPs capazes de fornecer aquele tipo de informação à APP.

A APP envia, então, ao AE uma função, chamada de função de utilidade (*utility function*) que, com base em um índice QoC (*Quality of Context*) de um CP, mensura o grau de satisfação da APP com o CP. Assim, para cada CP encontrado pelo SD, o AE carrega seu QoC e calcula a função de utilidade da APP solicitante. O CP que obtiver maior resultado no cálculo da função, é o escolhido para atender à APP. Quando escolhido o CP, o AE associa o CP ao CS daquele tipo de contexto e o liga à APP.

A partir deste ponto, toda necessidade de informações de contexto da APP será solicitada ao CS, que repassa a demanda ao CP selecionado, que interage

com os sensores necessários, compila os dados, gera a resposta ao CS e que a repassa para a APP.

Graças então a esta extratificação entre CPs, CSs e APPs, consegue-se o desacoplamento requerido à adaptabilidade de serviços. Se não fossem os CSs, as APPs seriam dependentes dos CPs utilizados, inviabilizando assim suas potenciais chances de adaptação.

4.3.2 Fouial, Fadel e Demeure

Já Fouial, Fadel e Demeure [11] possuem um trabalho para o provimento de serviços adaptativos em ambientes de computação móvel cujo enfoque já é intensificado à arquitetura para a adaptabilidade de serviços, e não ao processo de adaptação automática.

Por ela, três motivações justificam a necessidade de adaptação de serviços: mudanças das preferências do usuário, mudanças nas características do terminal de execução do serviço e mudanças na localização do serviço.

Em sua arquitetura, dispositivos móveis estão ligados a provedores de serviços (VASPs - *Value Added Service Providers*) através de operadores da rede móvel. Na operadora existe um gerenciador de serviços (VASM - *Value Added Service Manager*), capaz de localizar e selecionar serviços compatíveis com as características do terminal do usuário.

Assim, quando o usuário precisa de algum serviço, o dispositivo envia ao VASM a descrição de suas características de hardware e software. Com base nessas informações, o VASM devolve ao terminal móvel, uma interface visual para a localização de serviços, adequada às especificações do terminal. Através dessa interface, é possível ao usuário visualizar a lista de serviços registrados na operadora, classificados por categorias, acessar sua lista de favoritos e realizar consultas por serviços. Com a lista de serviços em mãos, o usuário seleciona o serviço de sua preferência que é, então, dinamicamente enviado ao dispositivo para execução.

Nesta proposta, a adaptapção é processada em um nível mais elevado que a anterior. Na abordagem da seção 4.3.1, a adaptação se dava no nível entre aplicações e provedores de serviço. Já em [11], a adaptação ocorre entre o usuário e os serviços, ou aplicações, que utiliza. Como os usuários são capazes de se adequarem a novas interfaces com maior facilidade que aplicações, não existe, neste modelo, qualquer elemento que abstraia os níveis de serviço. De uma forma ou de outra, permanecem os recursos de descrição, localização e recuperação de serviços, comuns em ambos os trabalhos.

4.3.3 Aura

Em [21], João Pedro Sousa e David Garlan, ao descreverem a camada *Environment Manager*, mencionam sua capacidade de monitorar o ambiente para a implementação de reconfiguração automática de recursos (aplicações e dispositivos) a fim de garantir a continuidade dos serviços já negociados com o módulo *Task Manager*. Em [50], define-se uma função de utilidade capaz de mensurar o quão boa é a capacidade de um recurso para atender às necessidades do usuário, com base no registro de suas preferências pessoais, mantidas pela *Task Manager*.

Assim, quando da falha de algum recurso ou da entrada de novos recursos no *smart-space*, cabe ao *Environment Manager* verificar, com base na função de utilidade, quais outras possíveis combinações de serviços podem melhor satisfazer o usuário. Ou seja, caso uma tarefa faça uso de um serviço de impressão que é provido por uma impressora X e, de repente, é disponibilizada no ambiente uma segunda impressora Y, com características superiores à primeira, o *Environment Manager* deve perceber esta mudança do meio, computar a função de utilidade da impressora Y, comparar o resultado com o da impressora X e, em sendo melhor, trocar, sem que o usuário perceba, o provimento do serviço da impressora X para a impressora Y.

Em [50] é dito que esta adaptabilidade de serviços, ou seja, esta facilidade do ambiente de promover a reconfiguração de recursos, é garantida graças a modelos que descrevem de forma abstrata a capacidade do *smart-space*. A grande virtude desta abordagem é o fato de a adaptação poder ser motivada tanto por mudanças nas intenções do usuário como por alterações no universo de recursos do ambiente.

4.3.4 Gaia

Em termos da adaptabilidade de serviços, o projeto Gaia foi motivado pela alta mobilidade dos ambientes ubíquos, que demanda das aplicações a necessidade de constante adaptação à realidade volátil de recursos em que roda. Ou seja, quando um usuário do sistema migra de um espaço para outro, todas as ligações e dependências que ele tinha com os dispositivos e serviços do espaço de origem devem ser reestabelecidas no novo ambiente, para a manutenção do funcionamento das aplicações que permaneceram em execução durante a transição.

Por [24], esta facilidade de adaptação, ou reconfiguração de serviços, até pode ser vista em outros projetos, mas sempre de uma forma restrita a situações controladas, com aplicabilidades reduzidas a domínios de problemas específicos. Raras as vezes se vêem propostas abrangentes e genéricas o bastante para serem utilizadas em larga escala, independente dos casos de utilização. Sob esta luz que

o projeto Gaia justifica seus esforços na definição de modelos abstratos para a integração de dispositivos e serviços heterogêneos, como o *Unified Object Bus*, a fim de promover o fácil intercâmbio de serviços e informações em um ambiente tão diverso quanto o da computação ubíqua.

4.3.5 GatorTech

As publicações do projeto não trazem referências explícitas a quaisquer recursos para a adaptação de serviços. Entretanto, na composição das camadas *Physical Layer*, *Sensor Platform Layer*, *Service Layer* e *Application Layer* transparecem as mesmas virtudes observadas no projeto EasyLiving, analisadas na seção 4.3.7.

4.3.6 ISAM

O projeto ISAM traz a adaptação de serviços como tema central de suas pesquisas. Tanto a arquitetura modular e multi-camadas do projeto, quanto a taxonomia de aplicações móveis definida em seu contexto são orientadas ao comportamento adaptativo de aplicações.

No entanto, a proposta do projeto é ampla e genérica. Conforme dito em [5], o mecanismo de adaptação é colaborativo, multi-nível e independente de domínio de problema. Ou seja, os cenários utilizados para ilustração da arquitetura referem-se desde à adaptação do *middleware*, causada por problemas de rede, à adaptação dos dados fornecidos por uma aplicação, devido à fatores pessoais, espaciais ou temporais. Quer dizer, a adaptação de serviços que, conforme definida na seção 4.1, trata apenas da troca de um provedor de serviços por parte de um cliente, é vista no projeto ISAM sob uma ótica muito mais abrangente e complexa. Por consequência, não foi possível identificar, dentro do projeto, os detalhes específicos dos mecanismos implementadores da adaptabilidade de serviços, conforme semântica aqui utilizada, apesar da certeza de sua existência.

4.3.7 EasyLiving

Graças ao desacoplamento entre a especificação e a implementação de serviços, o projeto EasyLiving traz consigo grandes colaborações para as técnicas de adaptabilidade de serviços como, segundo [31], a descrição abstrata de serviços com arquivos XML.

O grande objetivo desta abordagem é permitir que em um espaço que possua vários provedores de um mesmo serviço, exista a possibilidade de decisão do sis-

tema sobre a melhor alternativa para o provimento das funcionalidades requeridas ao atendimento de suas necessidades. Por exemplo, em um ambiente inteligente onde encontram-se vários dispositivos fornecedores do serviço de ponteiro, como *trackball*, *mouse* e tecnologias para a identificação de gestos manuais, seja possível a uma aplicação decidir qual a melhor ferramenta para assumir o controle de sua interface gráfica (o *trackball*, o *mouse* ou gestos manuais).

Em resumo, apesar de nem sempre ser tratada de forma explícita, a adaptabilidade de serviços é uma característica presente nos projetos de maior relevância da computação ubíqua. No capítulo que segue, sua importância para a *ubicomp* será estudada e, na sequência, uma proposta de arquitetura é sugerida para sua viabilização explícita e sistemática.

Capítulo 5

Adaptabilidade de Serviços contra o Desperdício de Recursos

"Ideally, users should not have to carry a machine around, just as people don't have to carry their own chairs."

João Pedro Sousa et. al., 2005

Este capítulo revisa o modelo de funcionamento da computação ubíqua, apresentando dois cenários didáticos, dos quais conclui-se a possibilidade de um potencial desperdício de recursos do *smart-space*. É feito então um paralelo deste problema com os desafios originais da computação em grade e a adaptabilidade de serviços é sugerida como estratégia para otimizar a utilitação de ambientes de computação ubíqua através do compartilhamento de recursos. Uma descrição desta proposta é apresentada ao final do capítulo.

5.1 Desperdício de recursos

A computação ubíqua caracteriza-se pela intensa especialização do computador, outrora de uso genérico, tornando-o peça ordinária do cotidiano humano de uso preciso e pontual. O princípio da invisibilidade, antes de abordagem futurística, diz respeito ao esforço da comunidade científica no sentido de integrar o processamento de dados a objetos comuns da vida humana, como copos, canetas eletrodomésticos, mantendo, ainda assim, a simplicidade de sua utilização. Neste pensamento, convergem pesquisas de diversos campos do conhecimento interessados na contextualização do homem no mundo em que vive, em busca da verdadeira inclusão digital.

Em essência, constituída por elementos da computação pervasiva e da computação móvel (figura 2.1), esta nova forma de computação utiliza-se da intensa

integração da informática com o mundo (seção 2.2.1) para a viabilização de ambientes inteligentes (*smart-spaces*) capazes de melhor servir à comunidade.

Smart-spaces dizem respeito à pulverização de pequenos elementos computacionais em locais naturalmente freqüentados pelo homem, como salas de reuniões, garagem ou escritório. Estes elementos, integrados entre si através de uma infra-estrutura de software e hardware invisíveis ao olho humano, captam sensações do ambiente que formatam o contexto (seção 2.2.4) pelo qual é orquestrada toda a oferta de serviços do lugar.

Nestes ambientes, homem e máquina deslocam-se com freqüência. O homem, imerso numa maré tecnológica, em simbiose perfeita com os dispositivos que o rodeiam e com uma infra-estrutura robusta de serviços, promove no *smart-space* um ambiente colaborativo onde dados fluem intensamente. Dos dispositivos partem infomações rumo ao usuário para o atendimento de suas necessidades. Dos usuários são capturadas ações que retroalimentam o sistema para adequação de seu comportamento às demandas do instante. A infra-estrutura de serviços, ou seja, o *middleware* deste sistema ubíquo, estabelece o canal de comunicação e a administração de todos os recursos físicos e lógicos necessários para esta dinâmica.

5.1.1 Casos otimistas de utilização

Apesar deste complexo universo cibernético, a usuários deste modelo computacional não deve ser exigido conhecimento da complexidade interna do ambiente.

Pelas premissas básicas da computação ubíqua, os dados de interesse de seus usuários devem estar disponíveis a qualquer hora e lugar, tornando o acesso à informação prática alheia às limitações de espaço e tempo enfrentadas nas arquiteturas tradicionais. Nesta realidade ideal, extiguem-se os dispositivos portáteis de armazenamento de dados (e.g. disco flexível, CDROM, *pen drive*), pois a informação tende a estar sempre ao alcance daqueles que a deseja, fato não observado nos paradigmas atuais, que obriga o indivíduo a sempre deslocar-se com as informações de que necessida, transportando-as a todas as ocasiões sociais que as demandem.

Sejam os exemplos a seguir:

> *Exemplo 1: É véspera de defesa de sua dissertação de mestrado e Luciana trabalha em seus slides de apresentação. Pela manhã, antes de sair de casa para a universidade, a estudante, insegura com a disponibilidade de conexão de rede do departamento, persiste seus arquivos em mídias portáteis para o transporte das informações de que necessita até a universidade.*

Exemplo 2: Para um professor participar da reunião do colegiado de seu departamento, é prudente que, antes, ele reveja toda a pauta da discussão a fim de munir-se de possíveis documentos que possam lhe ser úteis na defesa de suas opiniões.

Em ambos os exemplos, observa-se a constante previsão do usuário de informática a respeito dos dados que lhes serão necessários a cada próximo instante, e a necessidade permanente de manipulação de mecanismos complexos (CDROM, disquete, *pen-drive*) frente à promessa de transparência tecnológica da *ubicomp*. Orientada pela bandeira da simplicidade, a computação ubíqua tem a importante missão de extirpar do cotidiano todo os antagonistas da realidade humana, obrigando a computação a se adaptar da melhor forma possível ao mundo real, em oposição à dinâmica atual de adequação incisiva do mundo às inovações tecnológicas do amanhã.

Neste sentido, no mundo ideal previsto, observariam-se as realidades que seguem:

É véspera de defesa de sua dissertação de mestrado, Luciana trabalha em seus slides de apresentação e utiliza, repetidas vezes, a câmera web e o microfone de seu computador pessoal para o registro de seus ensaios da defesa. Pela manhã, seu despertador, integrado ao sistema meteorológico da cidade, verifica más previsões climáticas e antecipa-se, a fim de evitar possíveis atrasos da estudante. Ao sair de casa para a universidade, a mestranda dirige-se seguramente ao local da defesa. Ainda no caminho, solicita ao sistema de som do carro que reproduza o podcast *resultante de suas últimas gravações, na noite anterior. No campus, enquanto caminha rumo ao departamento, Luciana revisa os slides pelo celular. Ao entrar no auditório do departamento, a apresentação é carregada no sistema de projeção do ambiente. Para afastar o nervosismo enquanto aguarda a chegada dos membros da banca examinadora e convidados, Luciana recorre novamente ao seu celular para visualização de* traillers *dos últimos lançamentos do cinema, que obtivera da Internet*

Na descrição acima, Luciana participa de quatro *smart-spaces*: sua casa, seu carro, sua universidade e seu departamento. A informação relevante a Luciana naquele contexto, os slides de sua defesa, é onipresente. Em momento algum houve a necessidade de Luciana armazenar seus documentos de interesse em qualquer mídia móvel, e tampouco carregá-la consigo, conectá-la ao carro, montá-la no sistema de arquivos, desmontar, desconectar, conectar ao celular, montar, carregar, desmontar, desconectar, conectar na sala de apresentações, montar, carregar, etc. Todas as atividades foram realizadas de forma transparente pela infra-estrutura

do ambiente, pois os dados sempre estiveram disponíveis e toda a complexidade computacional de acesso à informação foi encapsulada pelos *middlewares* de cada *smart-space* pelo qual Luciana atravessou. O sistema de sua casa, conectado ao mundo por uma conexão de alta velocidade, integra simples objetos do dia-a-dia (e.g. despertador) a serviços WEB; O carro, conectado a satélites, acessa de forma transparente a informação produzida em casa; O celular, utilizando a rede de dados da operadora de telefonia móvel, recupera os slides da apresentação que, posteriormente, é transferida ao projetor do auditório para que este assuma a reponsabilidade de apresentação. Graças a esta conectividade, Luciana teve conforto, segurança e tranquilidade garantidos para a boa condução dos últimos instantes precedentes à sua defesa.

5.1.2 Casos pessimistas de utilização

A pervasividade da computação ubíqua, por natureza é resultado da integração de vários elementos de hardware e software com o meio em que residem. Luciana, na situação imaginária descrita acima, utilizou recursos que certamente não eram os únicos dos *smart-space* em que esteve. O ecosistema de dispositivos [51] que forma os ambientes ubíquos pelos quais Luciana passou, além do despertador (casa), do aparelho de som (carro), do celular (campus) e do projetor (departamento), é também composto por outros elementos provavelmente ignorados pela estudante. Considere a lista de possíveis elementos não considerados no exemplo:

1. Em casa:
    - Telefone móvel celular
    - TV digital
    - Refrigerador sensível a etiquetas RFID
    - Computador portátil com suporte à redes Bluetooth e WiFi
    - Câmera filmadora digital
2. No carro:
    - Telefone móvel celular
    - Conectividade WiMAX ou GPRS
    - Sistema de navegação GPS
    - Painel de controles customizável e sensível ao toque

- Reprodutor de áudio em formato digital
- Sistema de comandos por voz

3. No campus:
   - Telefone móvel celular
   - Rede sem-fio padrão 802.11
   - Sensores de presença
   - Leitores de impressão digital
4. No auditório:
   - Telefone móvel celular
   - Rede sem-fio padrão 802.11
   - Sistema de reprodução de áudio
   - Sistema de projeção de vídeo
   - Sensores de presença nas poltronas

Apesar da sinergia das atitudes de Luciana com as propostas da computação ubíqua, algumas possibilidades foram ignoradas:

1. Os vídeos gravados pela acadêmica nos ensaios na noite de véspera, através da webcam, poderiam ter sido gerados com péssimas qualidade de som e imagem, inviabilizando sua posterior visualização;
2. Durante o trajeto de casa à universidade, o telefone de Luciana poderia ter tocado, obrigando-a a parar o carro para atender a chamada, expondo-lhe ao risco de atraso;
3. A latência da rede GPRS da operadora, utilizada como canal de dados por Luciana no campus, poderia ter inviabilizado a completa carga dos slides e impedido a revisão do material;
4. A baixa resolução de tela do celular poderia impossibilitar a leitura das legendas do filme que assistia enquanto aguardava o início da apresentação.

Diante de previsões pessimistas como estas, deve-se atentar para alguns fatos:

1. Mesmo com o risco de produzir vídeos inutilizáveis, por conseqüência do uso de equipamentos precários, havia ociosa em sua casa uma câmera filmadora digital capaz de produzir resultados de qualidade certamente superior;

2. A razão que obrigou Luciana a estacionar o carro para responder à chamada em curso foi a impossibilidade de utilizar o celular sem a necessidade das mãos, apesar de seu veículo ser equipado de microfones integrados ao potente sistema de som, que poderiam servir de interface de comunicação com o dispositivo;

3. Mesmo sob a precariedade das tecnologias celulares de segunda geração, o telefone móvel de Luciana não seria capaz de utilizar a conexão 802.11 disponível nas dependências do campus.

4. A espera pelos participantes de sua defesa poderia ser longa, o que logo obrigaria Luciana a rapidamente desistir do esforço de compreensão dos textos ilegíveis apresentados junto ao filme em execução, aumentando-lhe a ansiedade, ainda que o sistema de projeção do auditório, de alta qualidade gráfica, estivesse em absoluta ociosidade.

Estas observações demonstram como a computação ubíqua ainda se apresenta precária em relação à alocação otimizada de recursos computacionais, apesar de todos os méritos reservados à inovação tecnológica.

5.1.3 Desperdício de recursos

É alta a densidade de dispositivos em um ambiente ubíquo, mas nem sempre aproveita-se deste "congestionamento" em prol da satisfação integral dos usuários do ambiente. Em termos computacionais, observa-se que o sistema ubíquo dispõe de um largo conjunto de recursos, mas nem sempre oferece meios de acessá-los. Analogamente, pode-se comparar esta situação com um cenário onde um sistema operacional não possua acesso a toda a capacidade de processamento, toda memória disponível ou a todos os dispositivos de hardware existentes em um computador.

Entretando, tal deficiência de compartilhamento e alocação de recursos computacionais não é exclusividade de arquiteturas ubíquas. A chamada computação em grade também teve suas origens sobre esta preocupação. Em suas primeiras propostas, como registradas no livro *The Grid: Blueprint for a New Computing Infrastructure* [52], defende-se o acesso a recursos computacionais com a mesma simplicidade e universalidade que hoje é oferecida a energia elétrica. No tradicional artigo *The Anatomy of the Grid* [53], Ian Foster et al. diz:

> *"The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. (...) This sharing is, necessarily,*

> *highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO)."(Ian Foster et al.)*

A descrição de Foster para o real problema da computação em grade adequa-se com retidão ao problema de alocação de recursos identificado na computação ubíqua, onde as *organizações virtuais* da primeira correspondem aos *smart-spaces* da segunda. Se acaso a malha de dispositivos dos ambientes frequentados por Luciana estivesse toda acessível de forma coordenada e segura, todos os riscos dos casos pessimistas de utilização de *smart-spaces* descritos no tópico 5.1.2 seriam mitigados.

Então, ao que parece, o desperdício de recursos está diretamente associado à incapacidade do sistema em promover o compartilhamento mencionado por Foster. Mas, na *ubicomp*, abstraindo cada recurso do *smart-space* como um serviço lógico do ambiente, este compartilhamento só é possível caso exista a possibilidade de adaptação desses serviços, que é a realocação de serviços entre os diversos clientes do espaço, conforme visto no capítulo 4, cuja existência implica na necessidade de incorporação da adaptabilidade de serviços ao ambiente.

Deste raciocínio, conclui-se que a solução para o desperdício potencial de recursos de um ambiente de computação ubíqua pode ser possível com a adoção da adaptabilidade de serviços, através da concepção de uma arquitetura flexível e robusta, capaz de de viabilizar o rearranjo de serviços para a melhor satisfação de seus usuários, conforme descrição que segue.

5.2 Adaptabilidade de serviços como solução ao desperdício de recursos

Frente às fragilidades na gerência de recursos observadas nas seções anteriores, é proposta uma arquitetura de software sobre a qual viabiliza-se a construção de *smart-spaces* sensíveis à adaptabilidade de serviços entre disposisitvos.

Esta arquitetura, motivada pela proposta de gerência de recursos dos sistemas operacionais modernos, será chamada de **UbiquitOS**, numa alusão a um sistema operacional imaginário da computação ubíqua, especializado na abstração e na viabilização do acesso aos recursos providos pelos inúmeros dispositivos presentes em um *smart-space*, inspirado pela filosofia de sistema operacional ubíquo do projeto Gaia, da seção 3.2.

No sistema UbiquiOS, implementa-se mecanismos para a orquestração dinâmica dos dispositivos do ambiente, garantindo-lhes interoperabilidade e possibiliade de compartilhamento de serviços entre si.

5.2.1 Premissas

Todas as decisões arquiteturais tomadas na concepção do sistem UbiquitOS partem das seguintes premissas básicas:

- Dá-se o nome de *smart-space* a qualquer ambiente físico, restrito, dotado de serviço computacional com recursos teoricamente infinitos [1], capaz de sustentar a infra-estrutura mínima necessária à execução do sistema UbiquitOS.
- Serão considerados para os experimentos dispositivos e tecnologias amplamente disponíveis no mercado, nesta data, como, telefones celulares, PDAs e computadores portáteis com tecnologia IrDA [54] e Bluetooth [55].

5.2.2 Definições

Para facilitar a comunicação, faz-se necessária a definição de alguns termos:

***Smart-space*(ambiente)** - Recinto físico com infra-estrutura computacional para execução do sistema UbiquitOS.

Middleware - Camada de software para a abstração dos recursos computacionais nativos do *smart-space*. Neste caso, o sistema UbiquitOS.

Dispositivo - Aparelho eletrônico com poder computacional.

Fabricante - Empresa responsaável pela construção de dispositivos.

Serviço - Recurso computacional lógico mantido em um dispositivo e acessível por outros.

Desenvolvedor - Indivíduo responsável pela programação de um serviço.

Provedor - Dispositivo hospedeiro e fornecedor de um serviço.

Consumidor - Dispositivo usuário de um serviço provido por outro dispositivo.

[1] Por exemplo, um cluster de computadores ligados a uma fonte abundante de energia, com ampla memória de execução e vasta capacidade de armazenamento e transmissão de dados.

5.2.3 *Storyboard*

Para melhor compreensão do problema, foi elaborada a ilustração da figura 5.1 que demonstra o cenário de execução do sistema. Dois momentos distintos são representados: o momento onde o dispositivo chega ao *smart-space* e o momento onde algum dispositivo solicita a adaptação de algum de seus serviços, chamados respectivamente de *handshake* e *adaptation*.

5.2.4 *Handshake*

1. Usuário define sua visibilidade no *smart-space* e aproxima-se de seu perímetro de cobertura.
2. *Smart-space* percebe a presença do dispositivo.
3. *Smart-space* registra dispositivo em seu ecosistema.
4. *Smart-space* solicita ao dispositivo a lista de serviços disponíveis.
5. Dispositivo fornece a lista de serviços ao *smart-space*.
6. *Smart-space* publica os serviços disponibilizados a todos os integrantes do ecosistema de dispositivos do ambiente.

5.2.5 *Adaptation*

1. Dispositivo deseja substituir o provedor de algum dos serviços que está utilizando.
2. Dispositivo solicita ao *smart-space* a lista de alternativas de provedores do serviço selecionado.
3. *Smart-space* envia ao dispositivo uma lista de outros provedores para o serviço.
4. Dispositivo seleciona o provedor que melhor lhe convier.
5. *Smart-space* promove a liberação dos recursos do antigo provedor do serviço.
6. *Smart-space* estabelece a ligação do dispositivo com o novo provedor do serviço.

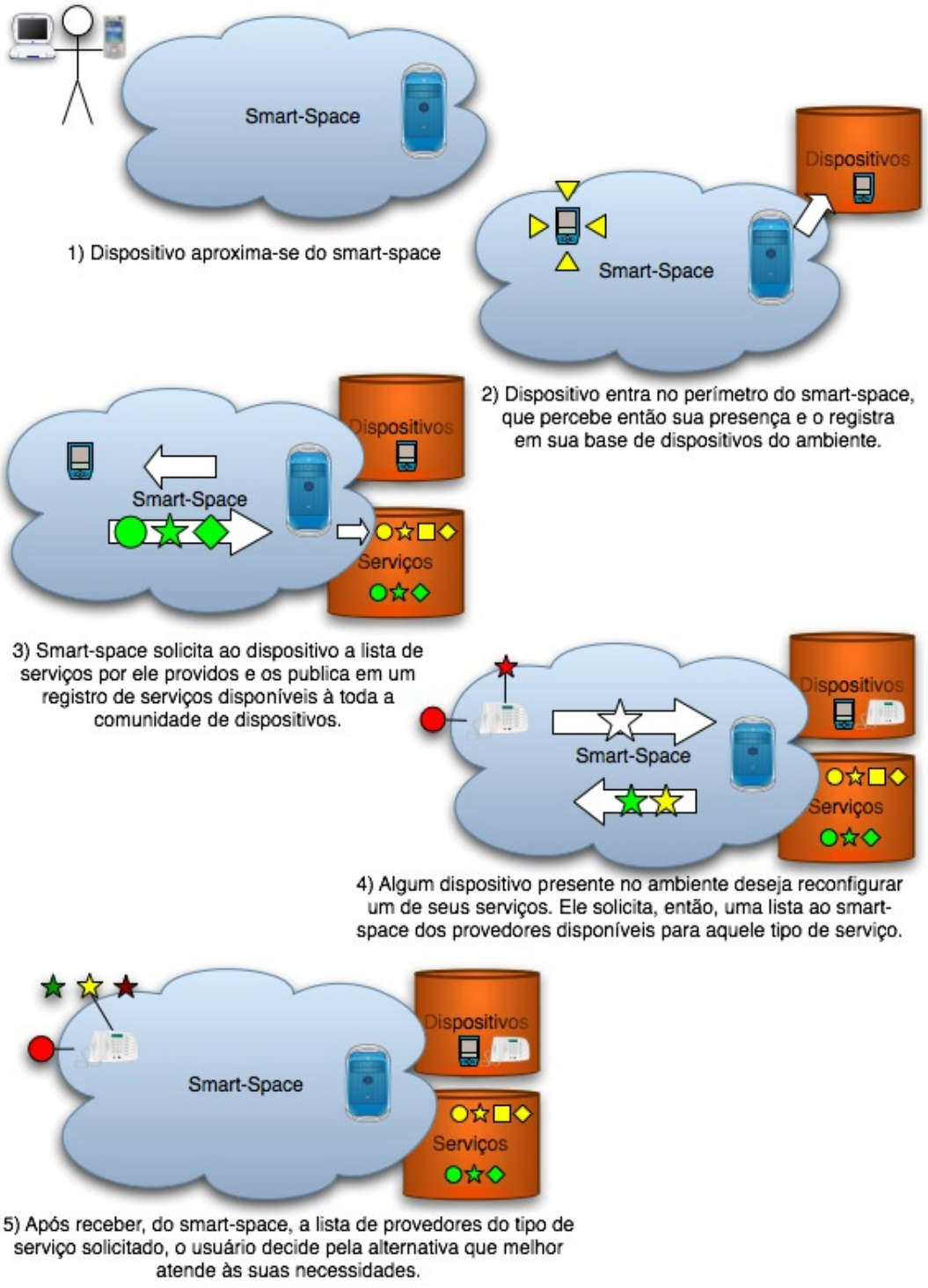

Figura 5.1: *Storyboard* do UbiquitOS

5.2.6 Decisões de projeto

Algumas questões sugiram durante o planejamento da implementação de cada cenário descrito no *storyboard*. As estratégias escolhidas foram influenciadas por fatores técnicos e gerenciais da proposta.

1. É realmente necessário o usuário do dispositivo configurar sua visibilidade no ambiente UbiquitOS?

   Caso não exista tal possibilidade de configuração, corre-se o risco de questionamentos sobre os limites entre a proposta da computação ubíqua, de enriquecimento da experiência do usuário com o meio, e a intrusão involuntária do sistema sobre as informações do usuário, preocupação também presente em [56] e [57].

   Concluída a relevância desta opção, há de se definir o mecanismo para o referido controle. A primeira opção seria associar a visibilidade pública ou privada do dispositivo com seu estado geral de funcionamento. Ou seja, dispositivos ligados, seriam considerados dispositivos publicamente visíveis, enquanto que dispositivos desligados, seriam dispositivos não visíveis na rede de cobertura do *smart-space*. Como ao usuário não seria dada a opção de utilizar o aparelho para outros fins, sem a intromissão do sistema UbiquitOS, descartou-se essa alternativa. A segunda opção seria através da associação da visibilidade do dispositivo com o estado de funcionamento de seu sistema de transmissão Bluetooth. Entretanto, por razões semelhantes às acima mencionadas, a opção também foi suspensa.

   Por fim, ficou definido que a visibilidade do dispositivo no ambiente UbiquitOS será configurada através da ativação, ou não, de um pequeno processo local, que garanta a disponibilidade de utilização do dispositivo para outros fins, ainda que utilizando a tecnologia Bluetooth de comunicação remota, sem a intromissão do sistema UbiquitOS.

2. Qual tecnologia utilizar para o sensoreamento de presença do dispositivo no perímetro do *smart-space*?

   Orientado pelas premissas da seção 5.2.1, a tecnologia de infra-vermelho seria pouco flexível, pois exigiria o alinhamento constante entre os dispositivos. Por esta razão, foi escolhida a tecnologia Bluetooth, por não exigir o direcionamento entre os dispositivos e, apesar do alcance médio de 10 metros, possuir cobertura proporcional à potência das antenas utilizadas.

3. Quem se responsabiliza pela percepção da entrada do dispositivo no *smart-space*?

   Caso o dispositivo seja o responsável por perceber sua entrada na área de cobertura do *smart-space*, seria-lhe obrigatório o constante monitoramento de sua localização, o que o levaria à maior necessidade de processamento e, consequentemente, ao aumento do consumo de energia.

   Por outro lado, caso o *smart-space* seja o responsável por tal monitoramento, por não ter limitações de consumo energético, pode manter a varredura constante do seu espaço de cobertura, a fim de perceber variações do conjunto de dispositivos que dele fazem parte. Esta segunda abordagem foi escolhida.

4. Qual tecnologia utilizar para a descrição dos serviços a serem publicados?

   Motivado pela tendência de mercado de ampla adoção das tecnologias de Web Services [58] para o desenvolvimento de serviços, seria natural a implementação de uma linguagem própria para a descrição dos serviços, talvez chamada de *Device Service Description Language*, aos moldes da *Web Service Description Language* [59].

   Todavia, frente à alta complexidade envolvida nesta tarefa, foram escolhidas simples interfaces Java, para a descrição dos serviços dos dispositivos, por sua simplicidade e facilidade de utilização no escopo proposto.

5. A descrição dos serviços a serem publicados residem no dispositivo ou no *smart-space*?

   Seria impossível manter no *smart-space* a descrição de todos os serviços de todos os modelos de dispositivos existentes, visto que a cada dia novos modelos de dispositivos podem ser lançados e novos serviços implementados.

   Desta forma, por influência da especificação USB, será adotado o modelo no qual o próprio dispositivo mantém a descrição dos serviços que provê, na forma de *drivers* de conexão ao serviço.

6. Qual tecnologia utilizar para publicação dos serviços do dispositivo no *smart-space*?

   Além da possibilidade de desenvolvimento de solução própria para publicação e manuteção de serviços no *smart-space*, várias tecnologias de mercado foram consideradas: RMI [60], CORBA [25], JXTA [61], Jini [62], OSGi [30] e UDDI [63].

Dentre todas, a tecnologia Jini chamou especial atenção por suas características de dinamicidade e espontaneidade da rede, essencialmente convergentes com as premissas básicas da computação ubíqua.

7. Como identificar provedores de um mesmo tipo de serviço?

   A primeira questão a ser analisada dizia respeito à necessidade ou não de se definir classes padrões de serviços, a serem seguidas por todos os implementadores. No caso de sua não existência, como os dispositivos consumidores de serviços poderiam descrever o tipo de serviço do qual necessitam? Do outro lado, como que dispositivos provedores de serviços poderiam descrever o propósito dos serviços que proviam?

   Frente a estas questões, concluiu-se a necessidade de haver a definição de categorias padronizadas de serviços, sob as quais cada desenvolvedor deveria enquadrar seus serviços. Através desta categorização, define-se um vocabulário comum pelo qual desenvolvedores, fabricantes, provedores e consumidores de serviços poderiam se benefiar diretamente através de maior clareza nas comunicações.

   Desta feita, outro desafio enfrentado dizia respeito à tecnologia a ser aplicada para a definição destas classes de serviços. Considerou-se a utilização de ontologias mas, pela necessidade de viabilização imediata da idéia, optou-se novamente pela utilização de interfaces Java.

5.2.7 Tecnologias

Com base nas decisões de projeto listadas na seção anterior e em outras definições tidas ao longo do projeto da arquitetura proposta, chegou-se à conclusão do seguinte parque de tecnologias a serem utilizadas como base do desenvolvimento do sistema UbiquitOS:

Java - Por sua característica multiplataforma e sua ampla base de dispositivos que lhe dão suporte, foi escolhida a Tecnologia Java como plataforma básica para todo o desenvolvimento.

Bluetooth [66] - Escolhida para ser a primeira tecnologia suportada para a comunicação entre dispositivos presentes no *smart-space*, por seu baixo custo, disponibilidade de bibliotecas de desenvolvimento e facilidade de utilização. Traz consigo recursos para percepção da presença de dispositivos, disponibilização e acesso remoto a serviços do dispositivo.

Jini - Tecnologia focada no princípio de que a rede não é um ambiente confiável e estável, características também observadas na computação ubíqua. Fornece uma infra-estrutura independente de protocolos para a construção de arquiteturas orientadas a serviços distribuídas e flexíveis. Escolhida como pilar do sistema de publicação e descoberta de serviços na rede do *smart-space*.

O capítulo que segue, descreve com detalhes o projeto de software desta proposta, firmado sobre as premissas, decisões e tecnologias vistas acima.

Capítulo 6

UbiquitOS

"With ubiquitous computing, using information technology will progressively feel more like using these everyday objects than using personal computers."

Andrew Fano and Anatole Gershman, 2002

A arquitetura UbiquitOS foi concebida com o objetivo inicial de viabilizar uma infra-estrutura de software para ambientes de computação ubíqua na qual a adaptação de serviços fosse facilitada. Sua descrição será feita numa abordagem *top-down*, na qual os módulos macros da arquitetura serão apresentados inicialmente e maiores níveis de detalhamento serão mencionados na sequência.

Com um modelo de implantação, busca-se apresentar as entidades principais que compõem um cenário funcional do sistema. No modelo de projeto, cada componente do modelo anterior terá suas estruturas internas e mecanismos detalhados, seguidos pela descrição dos protocolos básicos especificados para a integração de todas as peças vistas até então. Para finalizar, são apresentados o cenário de testes utilizado para validação da proposta e os resultados da pesquisa.

6.1 Modelo de Implantação

Um ambiente operacional do sistema UbiquitOS é constituído pelo *smart-space*, imóvel, dotado de um núcleo de software principal, e por vários dispositivos, geralmente móveis, que compartilham serviços entre si. Conforme mencionado na seção 5.2.6, toda a comunicação entre as partes da arquitetura será inicialmente realizada através da tecnologia Bluetooth.

Em resumo, três grandes módulos de software devem se fazer presentes para a constituição de um ambiente completo da arquitetura UbiquitOS, conforme ilustra a figura 6.1.

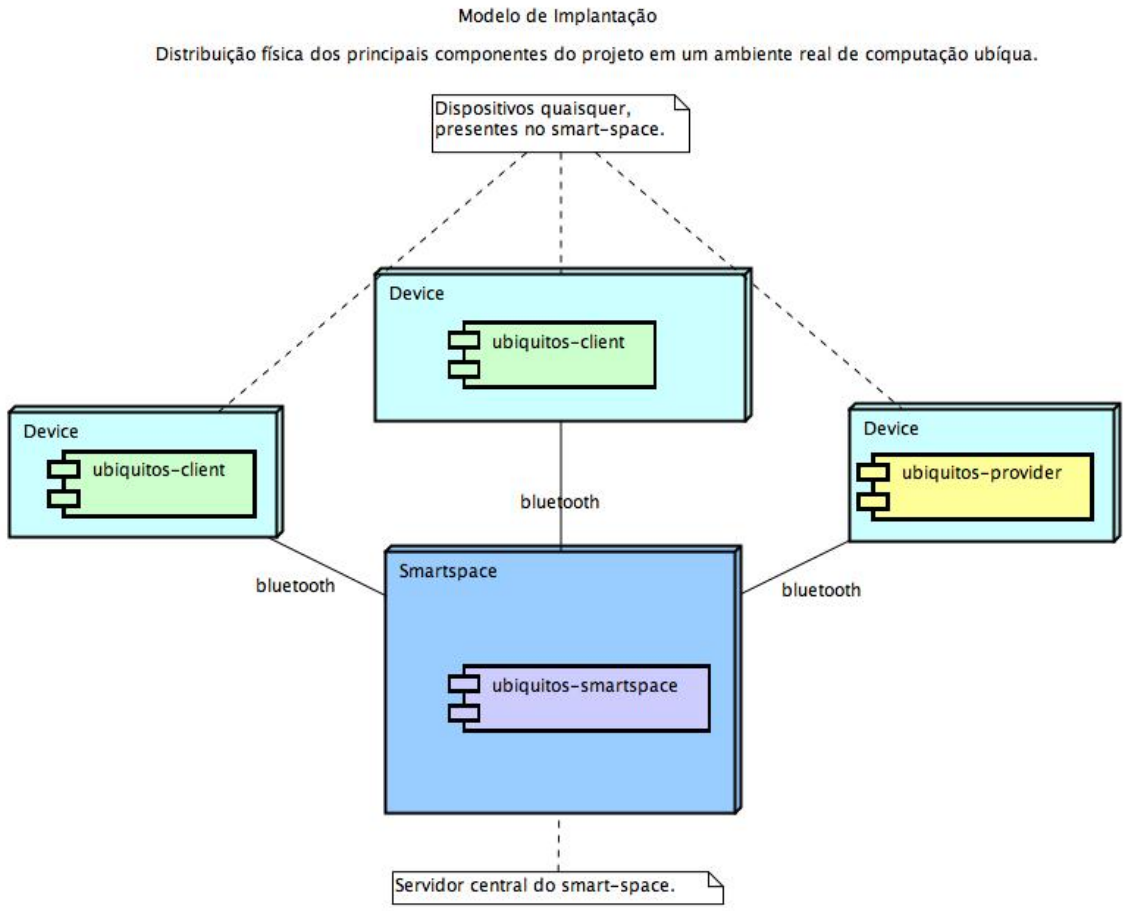

Figura 6.1: Modelo de Implantação

6.1.1 ubiquitos-smartspace

É o módulo da arquitetura UbiquitOS implantado no *smart-space*, em uma plataforma fixa e com ambundância de recursos computacionais.

Representa o núcleo do *smart-space*, implementando a camada de *middleware* necessária à abstração, orquestração e o controle de acesso a todos os serviços do ambiente.

É responsável pelo monitoramento constante do espaço para a percepção da entrada e saída de dispositivos, pela identificação dos dispositivos aderentes à especificação UbiquitOS, pelo primeiro contato com os dispositivos UbiquitOS, para publicação de seus serviços e por todo o mecanismo necessário à adaptação de serviços entre dispositivos.

6.1.2 ubiquitos-provider

É o módulo da arquitetura UbiquitOS implantado nos dispositivos com suporte à arquitetura e com capacidade de prover serviços a outros elementos da comunidade de dispositivos do ambiente.

Este módulo tem a responsabilidade de disponibilizar o acesso ao conjunto de serviços providos pelo dispositivo que o hospeda.

6.1.3 ubiquitos-client

É o módulo da arquitetura UbiquitOS implantado nos dispositivos com interesse em utilizar serviços providos por outros dispositivos.

Responsável por fornecer ao usuário a lista de provedores para determinado tipo de serviço e integrar-se ao dispositivo escolhido para a utilização de seus serviços.

6.2 Modelo de Projeto

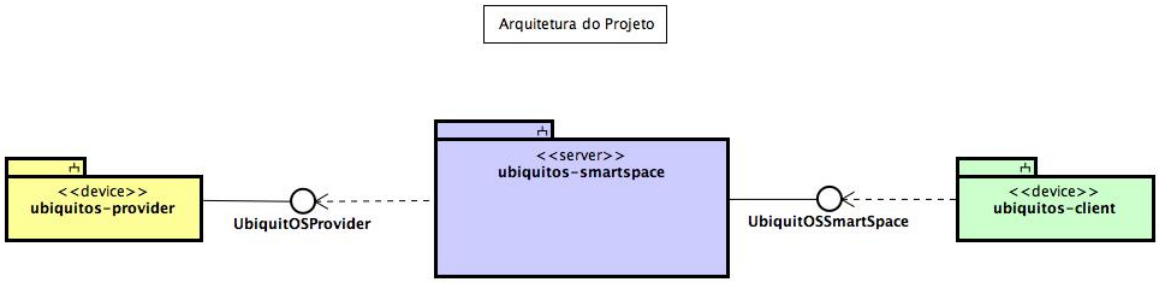

Figura 6.2: Arquitetura UbiquitOS

A figura 6.2 fornece um panorama geral da arquitetura do sistema UbiquitOS. A seguir serão descritos os detalhes de projeto de cada um de seus módulos. Para cada módulo, serão apresentadas suas visões estáticas e dinâmicas, através de diagramas da UML, para os cenários arquiteturalmente relevantes.

6.2.1 Módulos

Para dar apoio a todos os módulos, foram desenvolvidas duas biliotecas, BtUtil [64] e EasyJini [65], ambas disponibilizadas à comunidade sob licença livre, e com o objetivo de simplificar o acesso aos recursos das tecnologias Bluetooth e Jini, respectivamente.

6.2.1.1 ubiquitos-smartspace

O módulo `ubiquitos-smartspace` é constituído por cinco submódulos, ou subsistemas, vistos na figura 6.3, abaixo descritos:

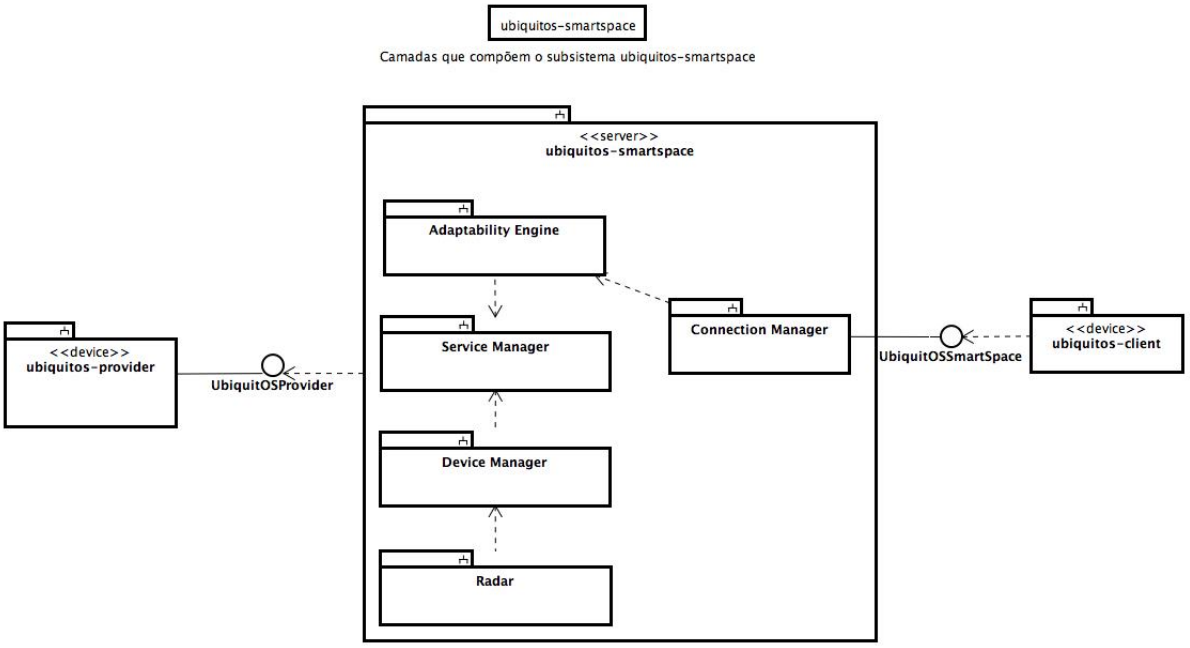

Figura 6.3: Camadas do módulo `ubiquitos-smartspace`

- **Radar**

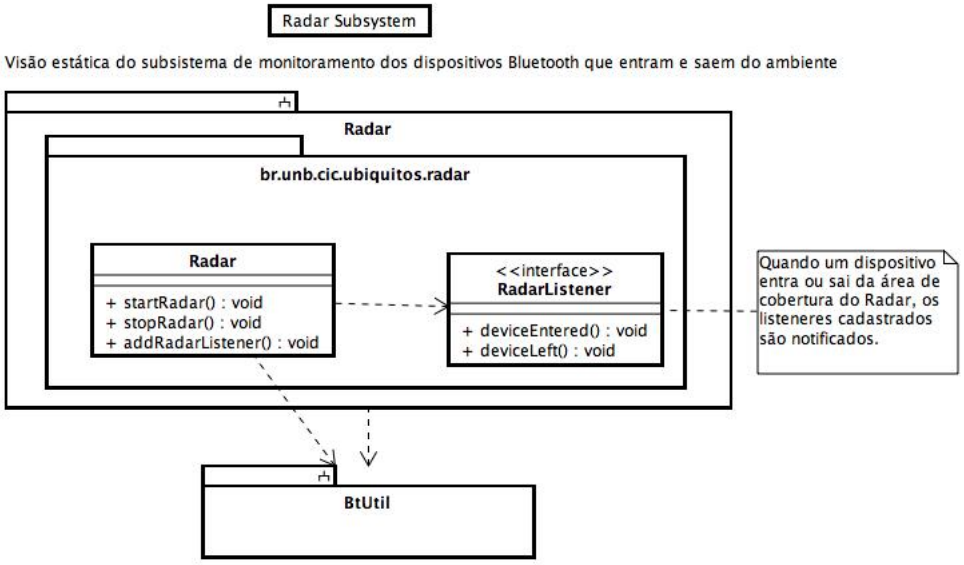

Figura 6.4: Subsistema Radar - Visão Estática

A camada mais básica do sistema, chamada Radar (figura 6.4), é responsável pela varredura constante do ambiente para a manutenção de uma lista dos dispositivos presentes [1] no perímetro de cobertura do sistema.

A base desta camada é a classe `Radar`, que se mantém em um fluxo contínuo de processamento para percepção da entrada e saída de dispositivos do ambiente. A cada novo elemento que entra ou sai do *smart-space*, uma notificação é emitida, através da interface `RadarListener`, aos objetos que previamente manifestaram tal interesse, pelo método `addRadarListener`.

Para sensoreamento do espaço, a classe `Radar` faz uso das facilidades da biblioteca BtUtil, que lhe permite a fácil inicialização de um *device inquiry* [66] e a recuperação das informações básicas de cada dispositivo localizado.

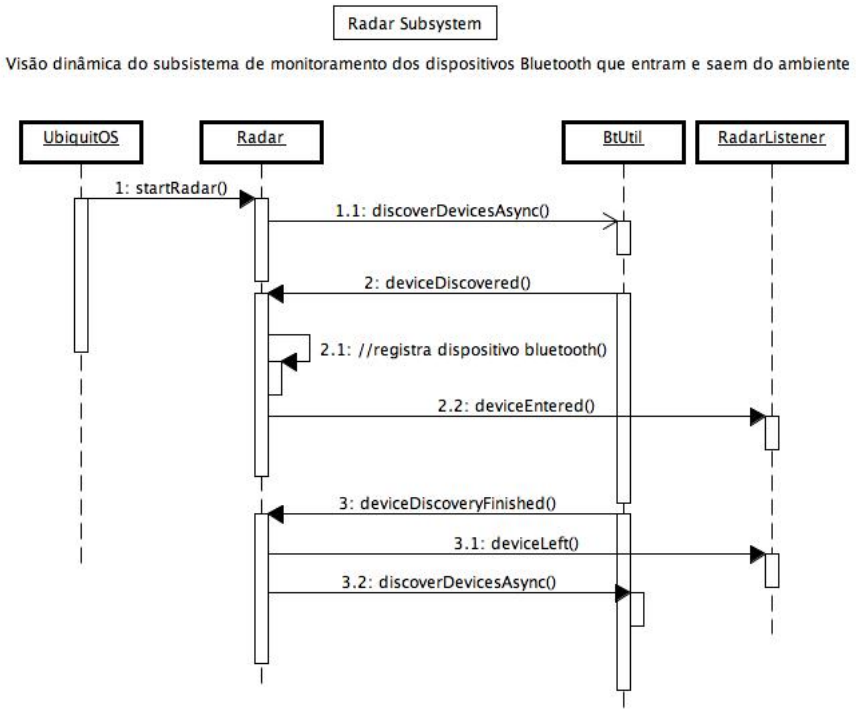

Figura 6.5: Subsistema Radar - Visão Dinâmica

Durante a inicialização do sistema UbiquitOS (figura 6.5), o subsistema Radar é ativado (mensagem `startApp`) e solicita então à BtUtil o início de um proceso assíncrono para a descoberta de dispositivos Bluetooth (mensagem `discoverDevicesAsync`). A cada dispositivo encontrado pela biblioteca, um evento é gerado e enviado ao subsistema Radar através do serviço

[1] Neste primeiro momento, a presença de dispositivos é baseada apenas na tecnologia Bluetooth.

`deviceDiscovered` de sua interface `BtUtilClientListener`. O Radar registra a referência do dispositivo Bluetooth localizado em sua base local e redireciona a notificação aos seus *listeneres* (interface `RadarListener`, mensagem `deviceEntered`). Ao final do processo de busca assíncrona por dispositivos Bluetooth, o Radar verifica se algum dos dispositivos previamente descobertos no ambiente não estão mais presentes no conjunto de dispositivos recém localizados e, em caso positivo, gera um novo evento a respeito (mensagem `deviceLeft`) e inicia, então, uma nova busca, permanecendo neste fluxo enquanto o módulo `ubiquitos-smartspace` estiver ativo.

- **Device Manager**

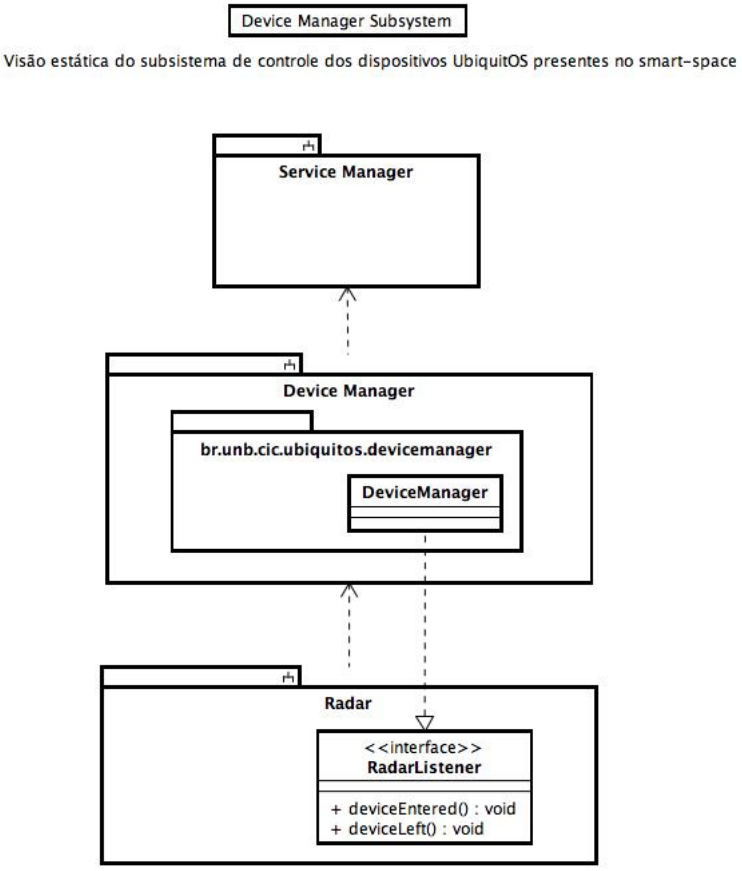

Figura 6.6: Subsistema Device Manager - Visão Estática

No próximo nível, encontra-se a camada Device Manager (figura 6.6), responsável pela manutenção do registro dos dispositivos presentes no *smart-space* que suportam a tecnologia UbiquitOS (não necessariamente todos os identificados pela camada Radar).

O núcleo deste subsistema é a classe `DeviceManager`, que implementa a interface `RadarListener` e se registra junto ao subsistema Radar para receber notificações sobre a entrada (`deviceEntered`) e saída (`deviceLeft`) de dispositivos do *smart-space*.

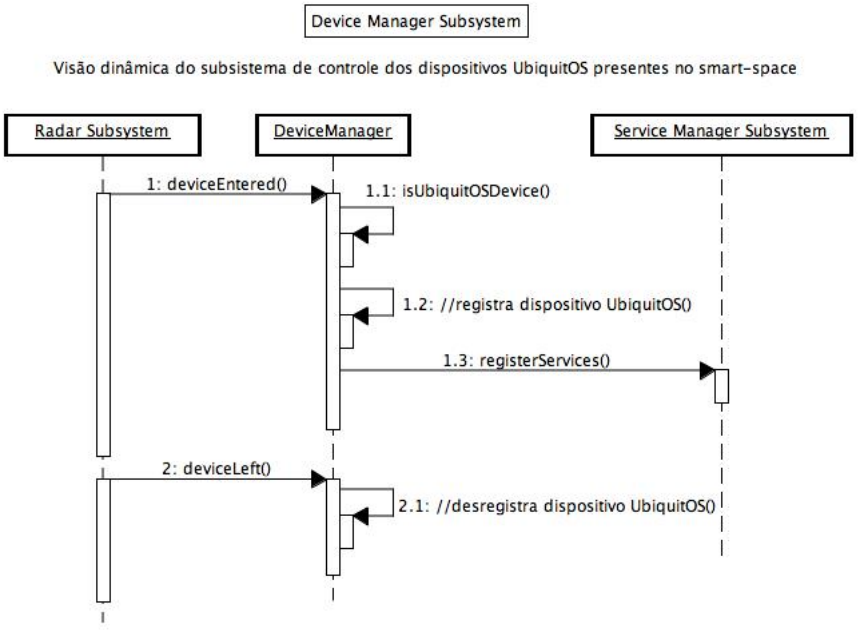

Figura 6.7: Subsistema Device Manager - Visão Dinâmica

Quando a classe `DeviceManager` recebe a notificação, do subsistema Radar, da descoberta de algum dispositivo Bluetooth no ambiente (figura 6.7), ela inicia o mecanismo para a publicação dos serviços do dispositivo no *smart-space*. Para o subsistema Device Manager, apenas dispositivos com suporte à tecnologia UbiquitOS lhe são de interesse. Logo, seu primeiro passo é executar esta verificação, através da mensagem `isUbiquitOSDevice`. Basicamente, esta operação verifica se o dispositivo é um provedor de serviços UbiquitOS, ou seja, se possui o módulo `ubiquitos-provider` da arquitetura.

Confirmada a compatibilidade do dispositivo com o sistema UbiquitOS, registra-se sua referência na base de dispositivos do Device Manager que, não necessariamente, é idêntica à base de dispositivos mantida pela camada Radar. Em seguida, solicita-se ao subsistema Service Manager a publicação dos serviços UbiquitOS providos pelo dispositivo recém descoberto e registrado.

- **Service Manager**

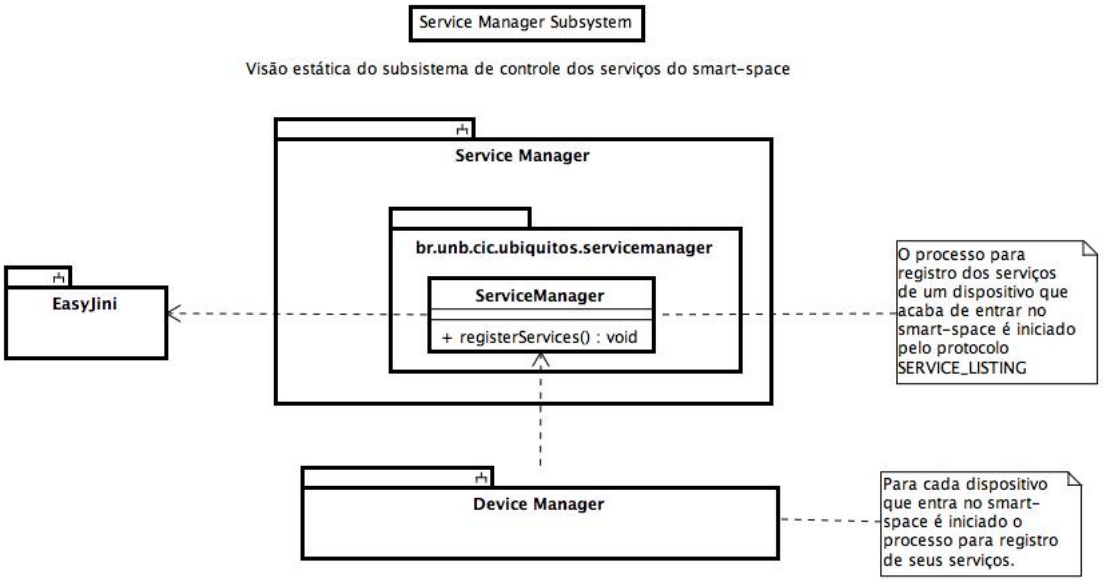

Figura 6.8: Subsistema Service Manager - Visão Estática

A camada Service Manager (figura 6.8) gerencia a publicação, classificação e descoberta dos serviços providos pelos dispositivos registrados na camada Device Manager.

A classe `ServiceManager` responsabiliza-se por dois processos: publicação e descoberta de serviços.

– Publicação de serviços

Consiste na publicação dos serviços disponíveis em cada dispositivo identificado pelo Device Manager, comunicando-se, através da biblioteca BtUtil, com o dispositivo para obtenção de sua lista de serviços e posterior publicação na rede, pela biblioteca EasyJini.

Conforme será visto na seção 6.2.1.2, cada dispositivo mantém, de forma serializada, os *drivers* necessários para comunicação e utilização de seus serviços. Cabe ao Service Manager a recuperação, deserialização, instanciação e publicação destes *drivers* no *smart-space* para que estejam disponíveis a toda comunidade de dispositivos do ambiente.

Na figura 6.9 , onde se vê a mensagem semântica `obtém lista de drivers`, compreende-se o estabelecimento de uma conexão Bluetooth com o dispositivo provedor de serviços, a leitura, byte a byte, da definição dos *drivers* de serviços lá armazenados de forma serializada e a deserialização, já no *smart-space*, dessas classes. Por fim, cada *driver*

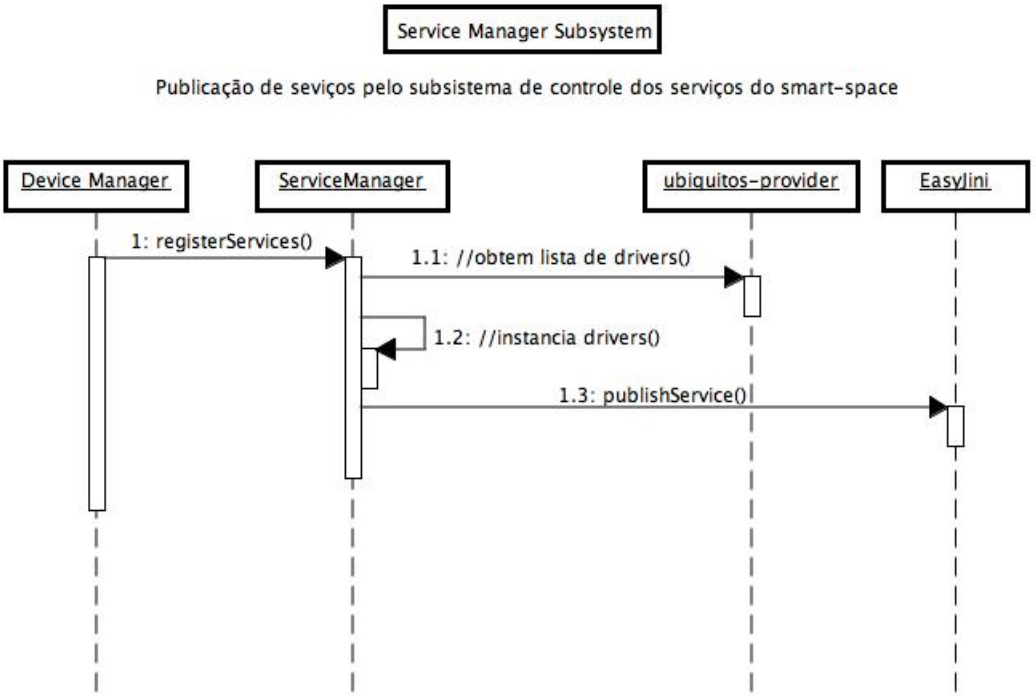

Figura 6.9: Subsistema Service Manager - Publicação de Serviços

é instanciado e seus objetos são publicados no diretório de serviços da arquitetura Jini, através da biblioteca de apoio EasyJini.

– Descoberta de serviços

A descoberta de serviços consiste na localização e recuperação de serviços já publicados, para que sejam utilizados por outros dispositivos do *smart-space*.

Iniciada quando um usuário deseja fazer a adaptação de um serviço, a solicitação da descoberta de serviços parte do Adaptability Engine, chega ao Service Manager e é delegada à biblioteca EasyJini, como ilustra a figura 6.10. Nesta primeira transação, busca-se uma lista de dispositivos capazes de prover o serviço desejado. Uma vez escolhido o dispositivo provedor do serviço requerido, uma segunda transação é inicializada, para a busca de seu objeto de acesso.

- **Connection Manager**

Este módulo funciona como fachada concentradora de todas as requisições recebidas pelo `ubiquitos-smartspace`. Na atual arquitetura, apenas o módulo `ubiquitos-client` é capaz de gerar requisições ao *smart-space*.

Em síntese, as demandas do módulo `ubiquitos-client` resumem-se

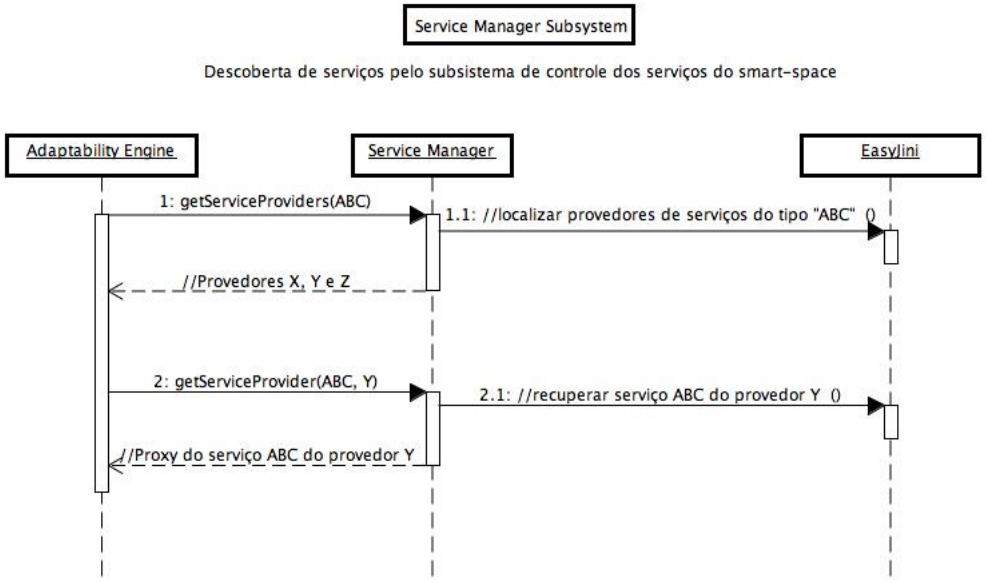

Figura 6.10: Subsistema Service Manager - Descoberta de Serviços

naquelas relacionadas à descoberta de serviços do Service Manager, da seção acima. A figura 6.11 detalha a interface de comunicação entre os módulos `ubiquitos-smartspace` e `ubiquitos-client`, implementada pelo Connection Manager, previamente apresentada na figura 6.2. A figura 6.12 apresenta as dependências estáticas deste com outros subsistemas do projeto e a figura 6.13 ilustra, sob a ótica do Connection Manager, os processos dinâmicos de localização e recuperação de serviços do *smart-space*.

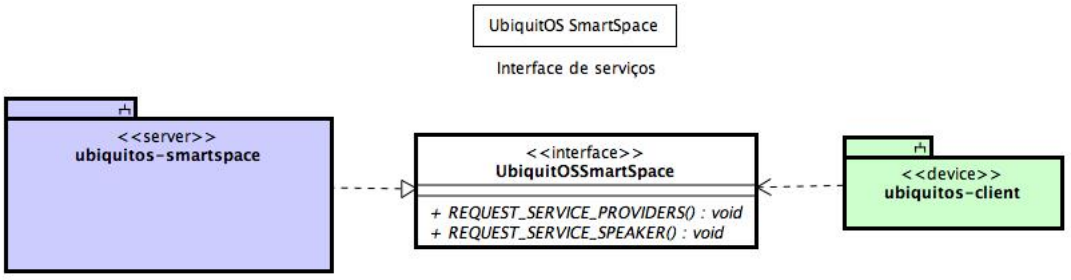

Figura 6.11: Interface de Serviços do Módulo `ubiquitos-smartspace` disponibilizada pelo Connection Manager

Para demonstração do comportamento dinâmico deste módulo, foi desenvolvido um pequeno e simples serviço, a ser utilizado como caso de teste da arquitetura. Trata-se do serviço *Speaker*, cuja única funcionalidade disponível é a capacidade de gerar uma saudação como "Bom Dia", "Olá", ou algo semelhante. Isso explica, nas figuras 6.11 e 6.13, a presença da mensa-

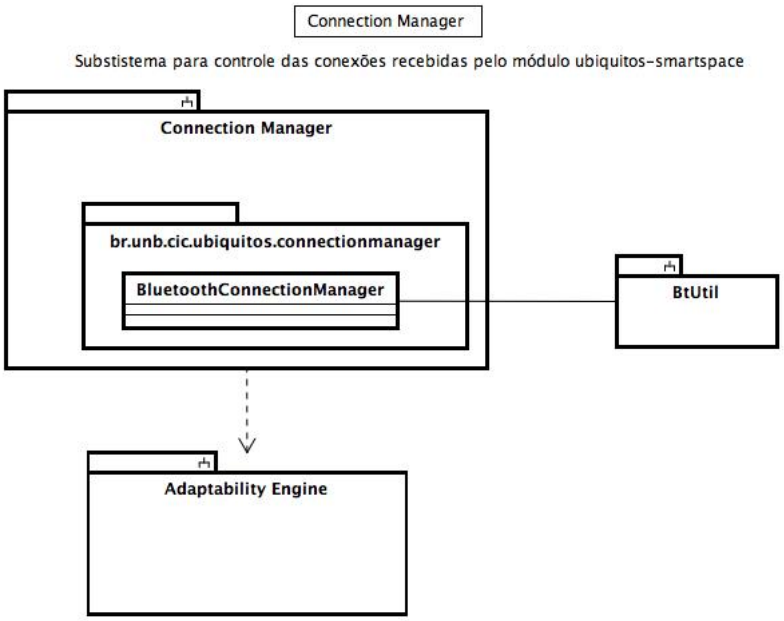

Figura 6.12: Subsistema Connection Manager - Visão Estática

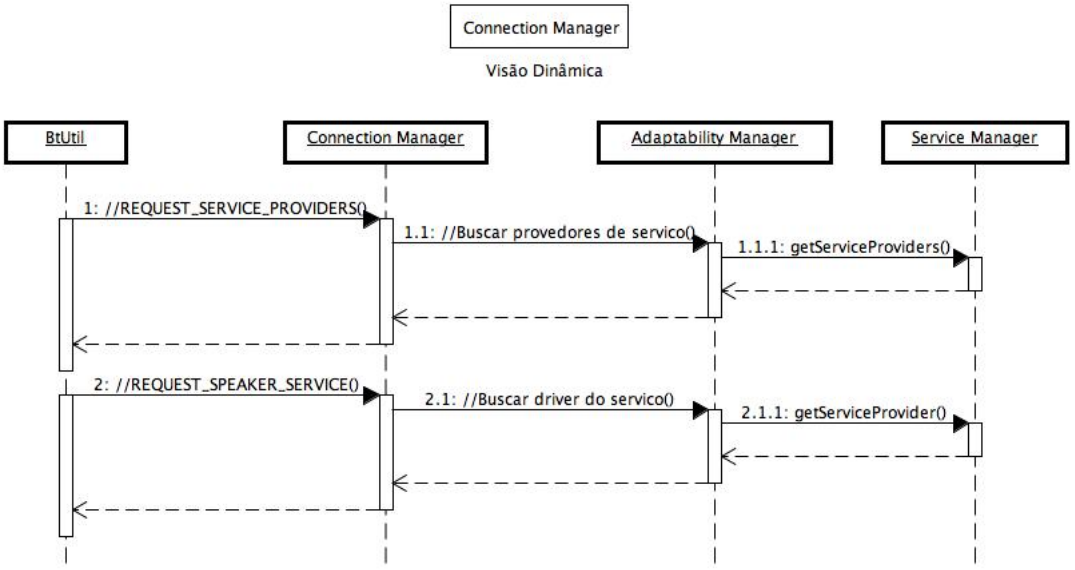

Figura 6.13: Subsistema Connection Manager - Visão Dinâmica

gem *REQUEST_SPEAKER_SERVICE*, que representa a requisição deste serviço específico.

- **Adaptability Engine**

  Mantém a responsabilidade de orquestrar o processo de adaptação de serviços (*adaptation*), descrito na seção 5.2.3. Como rascunhado nas seções

anteriores, este módulo depende do Connection Manager, para receber de clientes solicitações de adaptação de serviço, e do Service Manager, para viabilizar a descoberta de serviços na rede do *smart-space*, como mostra a figura 6.14.

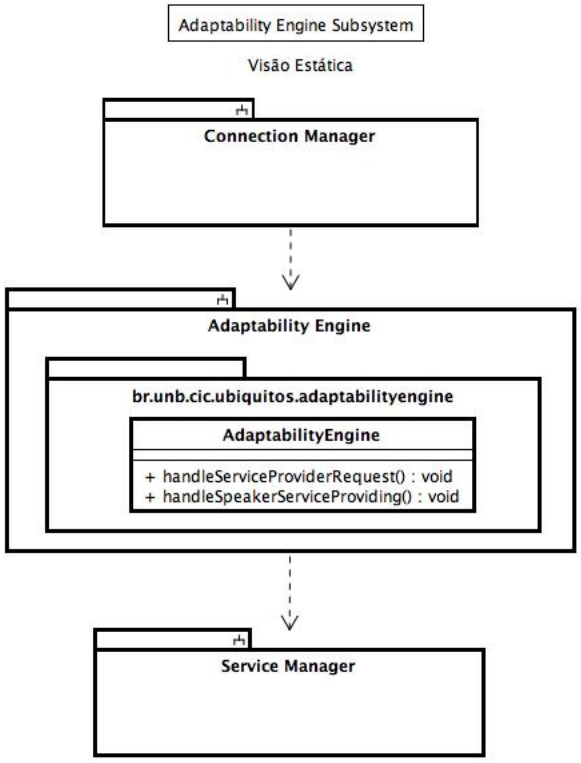

Figura 6.14: Subsistema Adaptability Engine - Visão Estática

Ao receber uma requisição para busca de provedores de um tipo de serviço qualquer, o módulo, com o auxílio do Service Manager, recupera uma lista de dispositivos capazes de prover o serviço solicitado e a envia de volta ao dispositivo cliente.

O usuário do dispositivo cliente seleciona um dos provedores da lista e então uma nova requisição é enviada ao Adaptability Engine que, novamente com o apoio do Service Manager, recupera, da rede, o serviço selecionado, ativando-o logo em seguida (figura 6.15).

O mecanismo de conectores utilizado para a comunicação entre cliente e serviço remoto é descrito na seção 6.2.2.1.

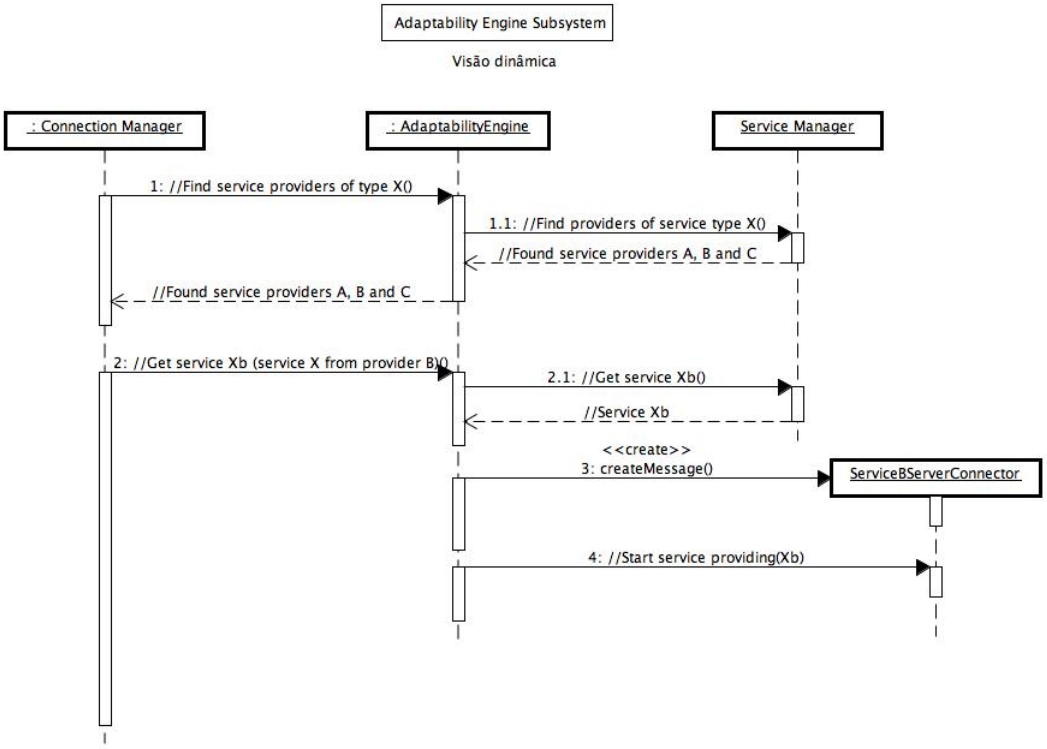

Figura 6.15: Subsistema Adaptability Engine - Visão Dinâmica

6.2.1.2 ubiquitos-provider

O módulo `ubiquitos-provider` responde pela disponibilização do acesso aos serviços capazes de serem providos por um dispositivo.

Um serviço provido por algum dispositivo só tem valor no *smart-space* se puder ser utilizado por outros dispositivos. Entretanto, cada dispositivo provedor pode implementar mecanismos próprios para acesso aos seus serviços, pois as tecnologias de comunicação e os protocolos suportados são distintos entre diferentes modelos e fabricantes. Por esta heterogeneidade, qualquer tentativa de generalização do esquema de acesso e serviços de dispositivos tende a ser um processo de extrema complexidade.

Por exemplo, um computador não se comunica espontaneamente com alguma recém lançada impressora. É preciso que antes o *driver* do referido periférico seja instalado no computador, e assim ocorre sempre que é necessária a comunicação entre dois dispositivos que se desconhecem. Normalmente, cada dispositivo provedor de serviço (e.g. impressora) traz consigo uma mídia na qual encontra-se o software necessário à sua utilização.

Um primeiro sinal de evolução surgiu com as arquiteturas *Plug and Play* que, em geral, caracterizam-se pela manutenção, do dispositivo cliente, de uma am-

pla lista de *drivers* potencialmente necessários, e cuja utilização se dará apenas quando da conexão de algum dos dispositivos suportados pela lista de *drivers*. O inconveniente desta abordagem é a necessidade de armazenamento de tantos *drivers* e a consequente possibilidade de não suporte a algum dispositivo recém lançado no mercado. E, tendo em vista a realidade da computação ubíqua, na qual os dispositivos clientes são plataformas com limitações computacionais extremas, tal estratégia de armazenamento de tamanha lista cai por terra.

Como resposta à este problema, o projeto *The Gator Tech House*, da seção 3.3, propõe uma técnica na qual o próprio dispositivo provedor de algum serviço mantém consigo os *drivers* necessários à sua utilização, que são transmitidos dinamicamente, aos interessados, conforme a necessidade. Nesta proposta, em oposição às tentativas de padronização do acesso ao serviço, busca-se a padronização do processo de transferência, instanciação e execução, pelos dispositivos clientes, dos *drivers* fornecidos pelos dispositivos provedores, estratégia adotada no projeto UbiquitOS.

Então, no módulo `ubiquitos-provider` são mantidos, de forma serializada, todos os *drivers* de acesso aos serviços providos pelo dispositivo. Quando o dispositivo entra na área de cobertura do *smart-space* e o módulo `ubiquitos-smartspace` do *smart-space* solicita ao módulo `ubiquitos-provider` do dispositivo sua lista de *drivers* disponíveis (seção 6.2.1.1), o dispositivo envia ao *middleware* todo o *bytecode* de *drivers* que mantém serializado. Um exemplo deste *bytecode*, serializado, de um *driver* de serviço, mantido pelo módulo `ubiquitos-provider`, pode ser apreciado na tabela 6.2.1.2. O *middleware*, na figura do módulo `ubiquitos-smartspace`, recebe este código, deserializa-o, instancia-o e o publica no registro de serviços do ambiente. Sobre a recuperação e utilização deste *driver*, vide seção 6.2.1.3.

```
0xCA, 0xFE, 0xBA, 0xBE, 0x00, 0x00, 0x00, 0x31, 0x00, 0x16, 0x00, 0x04,
0x00, 0x11, 0x08, 0x00, 0x12, 0x07, 0x00, 0x13, 0x07, 0x00, 0x14, 0x07,
0x00, 0x15, 0x01, 0x00, 0x06, 0x3C, 0x69, 0x6E, 0x69, 0x74, 0x3E, 0x01,
0x00, 0x03, 0x28, 0x29, 0x56, 0x01, 0x00, 0x04, 0x43, 0x6F, 0x64, 0x65,
0x01, 0x00, 0x0F, 0x4C, 0x69, 0x6E, 0x65, 0x4E, 0x75, 0x6D, 0x62, 0x65,
0x72, 0x54, 0x61, 0x62, 0x6C, 0x65, 0x01, 0x00, 0x12, 0x4C, 0x6F, 0x63,
(...)
0x12, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x01, 0x00, 0x00, 0x00,
0x03, 0x00, 0x0B, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x01, 0x00, 0x0F, 0x00,
0x00, 0x00, 0x02, 0x00, 0x10
```

Tabela 6.1: Exemplo de *driver* serializado no `ubiquitos-provider`

Existe uma implementação base do módulo `ubiquitos-provider` que deve ser

especializada por cada dispositivo com interesse na disponibilização de serviços para o *smart-space*, como pode ser visto na figura 6.16.

Já descrito na seção 5.2.6, a visibilidade do dispositivo no ambiente do sistema UbiquitOS deverá ser ativada manualmente, pelo usuário do dispositivo, através de um processo independente a ser executado no aparelho. Pela opção de plataforma de desenvolvimento feita, este processo para ativação da visibilidade do dispositivo como provedor de serviços no *smart-space* é implementado através de um pequeno aplicativo que, ao ser executado, inicializa os serviços necessários à comunicação do módulo `ubiquitos-smartspace` para a publicação dos *drivers* disponíveis, quando de entrada do dispositivo em sua área de cobertura.

A única responsabilidade do aplicativo (a classe de fronteira NokiaN80UbiquitOSProviderMidlet da figura 6.16) é o registro de cada *driver* existente na implementação básica do módulo (classe BaseServiceProvider), através do método `registerServiceDriver`, que recebe como parâmetro o nome do *driver*, para sua posterior instanciação, e seu código binário, em formato serializado.

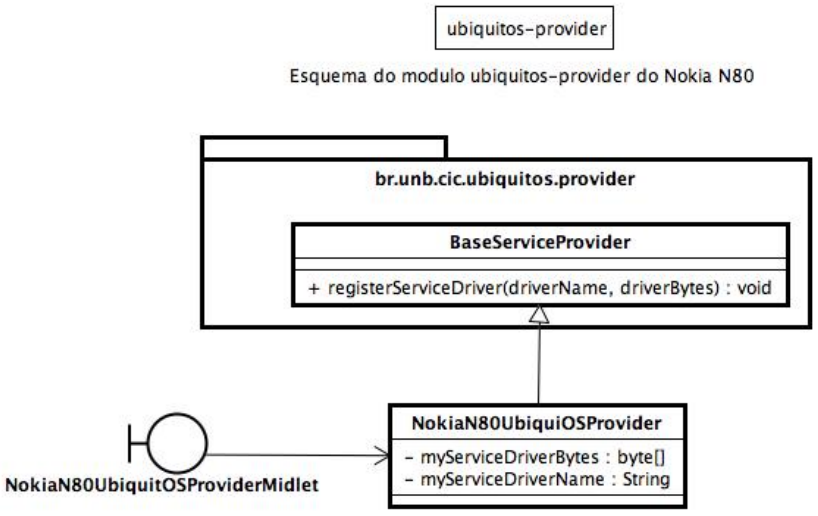

Figura 6.16: Utilização do módulo `ubiquitos-provider` para disponibilização de serviços para o ambiente

6.2.1.3 ubiquitos-client

O módulo `ubiquitos-client` responsabiliza-se pela viabilização da descoberta e seleção de provedores de serviços e pela recuperação e utilização do *driver* do serviço escolhido.

Uma aplicação com interesse em se beneficiar da adaptação de serviços, ou

seja, da utilização de serviços disponíveis no *smart-space*, deve fazer uso do módulo `ubiquitos-client`, conforme ilustrado pela figura 6.17.

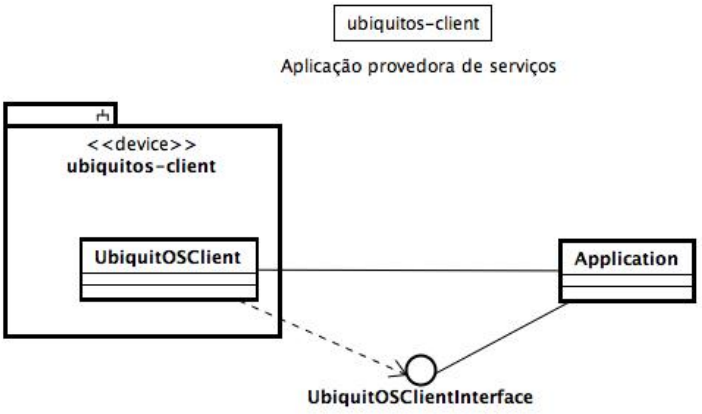

Figura 6.17: Utilização do módulo `ubiquitos-client` para usufruto da adaptação de serviços do ambiente

Internamente, o módulo é composto por um conjunto de três componentes que atuam nos papéis de fachada de serviços (UbiquitOSClient), interface de interação (UbiquitOSClientGUI) e regras de negócio (UbiquitOSClientController). A figura 6.19 descreve o processo em detalhes. No geral, toda solicitação provinda das aplicações são recebidas pela classe UbiquitOSClient que ativa a interface para a adaptação de serviços através da classe UbiquitOSClientGUI. Através desta interface, é apresentada ao usuário uma lista dos serviços do dispositivo passíveis de adaptação. Selecionado o serviço a ser adaptado, o usuário solicita, ainda através desta GUI, a lista de provedores existentes no *smart-space*, que é obtida do módulo `ubiquitos-smartspace` através da classe UbiquitOSClientController. Nesta lista, constam todos os dispositivos capazes de oferecer a funcionalidade em questão. Após o usuário escolher o provedor desejado, conforme suas preferências pessoais, novamente a classe UbiquitOSClientController busca o serviço remoto na rede, cujo acesso é então disponibilizado através de um mecanismo de conectores, descritos na seção 6.2.2.1, que possibilitam ao usuário utilizar serviços remotos com a mesma semântica de utilização do serviço local que outrora utilizava.

6.2.2 Mecanismos

Nesta seção são descritas as micro-estruturas de projeto concebidas para a resolução de problemas recorrentes da aplicação.

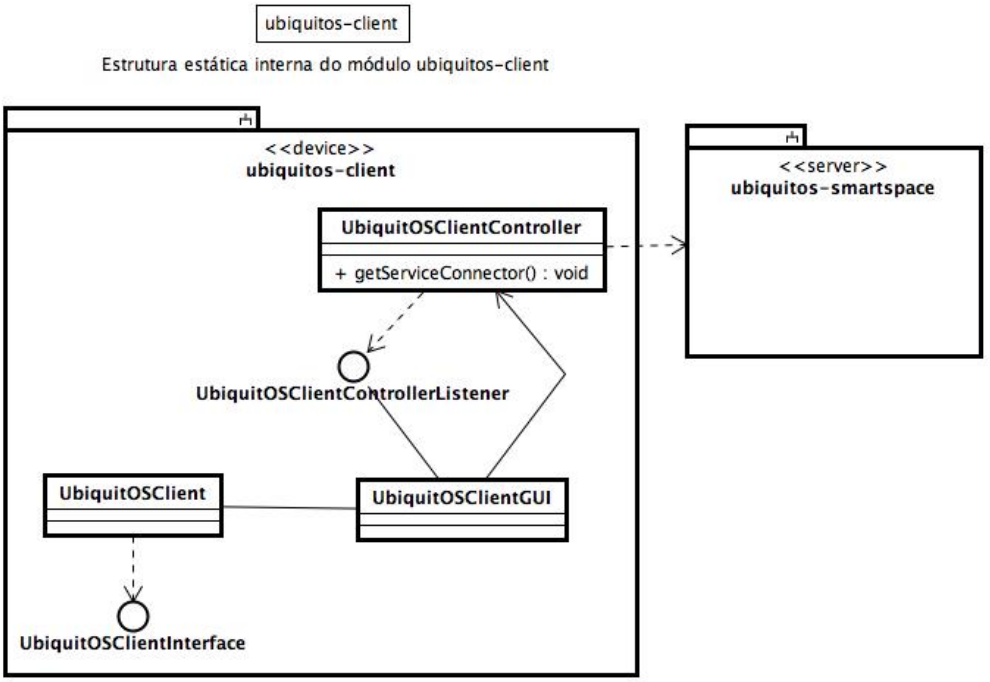

Figura 6.18: Estrutura interna do módulo `ubiquitos-client`

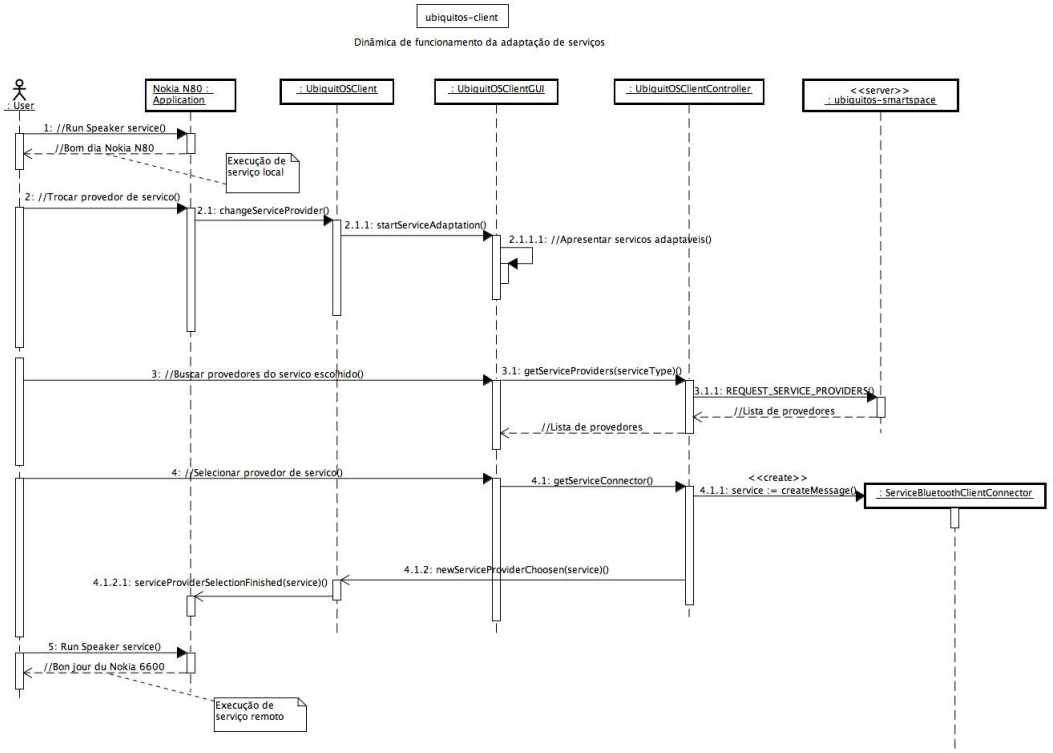

Figura 6.19: Adaptação de serviço através do módulo `ubiquitos-client`

6.2.2.1 Conectores

Ambos os diagramas de sequência representados nas figuras 6.15 e 6.19 são concluídos com classes chamadas de conectores, que se referem a um mecanismo arquitetural desenvolvido para superar as restrições de carga dinâmica de código das quais são vítimas os aplicativos desenvolvidos sobre a plataforma Java Micro Edition.

A seção 6.2.1.2 descreve a importância dos *drivers* de serviço para viabilização do seu acesso. Esses *drivers*, conforme visto na mesma seção, são mantidos em formato serializado nos dispositivos provedores de serviços e, quando da entrada em um *smart-space*, são enviados ao *middleware* do ambiente para publicação. Uma vez publicados na rede, os *drivers* lá permanecem até que sejam solicitados por algum dispositivo cliente interessado em utilizar seus respectivos serviços. Teoricamente, esse *drivers* deveriam ser enviados ao dispositivo cliente, onde seriam utilizados para acesso aos serviços remotos. Entretanto, por uma razão de segurança da tecnologia, a plataforma Java Micro Edition, utilizada para desenvolvimento do projeto, não permite a carga dinâmica de classes, inviabilizando assim o carregamento, em tempo de execução, do código binário dos *drivers* de serviço pelos dispositivos cliente, implicando a busca de estratégias alternativas para o alcance dos objetivos desejados. Assim, com base na proposta de [67], desenvolveu-se este mecanismo de conectores, que estabelecem a ponte entre dispositivo cliente e *driver*, eliminando a necessidade de sua transferência e carga dinâmica.

A figura 6.20 ilustra a estrutura deste mecanismo. Para cada classe de serviço (e.g. ServiceA) existe um par de conectores cliente (e.g. ServiceABluetoothClientConnector) e servidor (e.g. ServiceABluetoothServerConnector). Analogamente, estes elementos trabalham como os Stubs e Skeletons da tecnologia CORBA. Para a aplicação comunicar-se com o serviço remoto, é feita uma requisição local ao conector cliente, de igual interface ao serviço remoto, que se responsabiliza em formatar uma mensagem de rede (Bluetooth, no exemplo) que é então recebida pelo conector servidor, interpretada e transmitida ao *driver* do respectivo serviço, que trata, por sua vez, de reencaminhar a requisição ao serviço, aonde quer que esteja, e de processar a resposta, despachando-a no caminho inverso.

Dessa forma, apesar da impossibilidade de envio do *driver* diretamente do *smart-space* ao dispositivo cliente, pelas restrições mencionadas, criou-se uma ponte que estabelece o contato entre aplicação e *driver*, necessária ao alcance do serviço disponibilizado no ambiente.

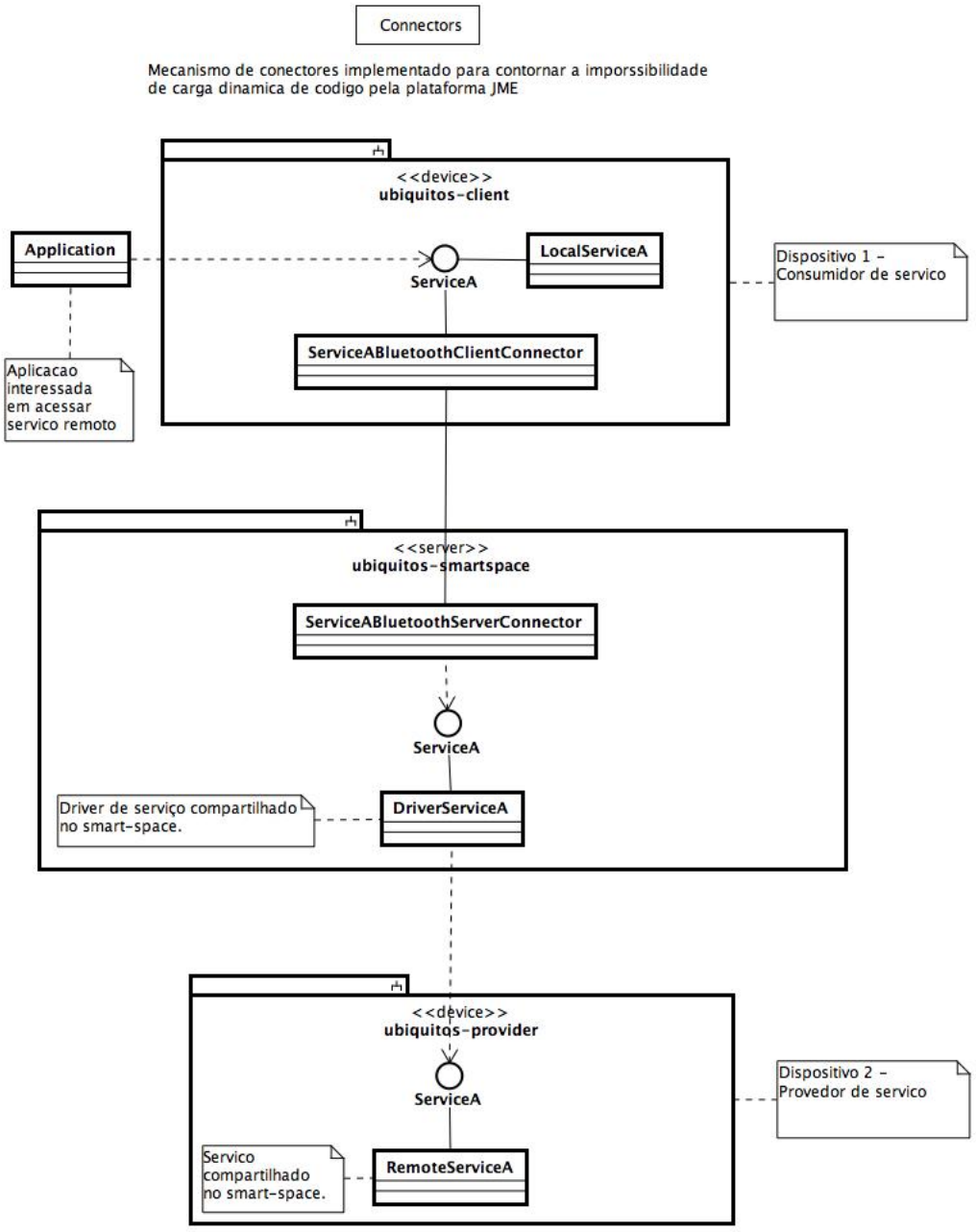

Figura 6.20: Mecanismo de conectores

6.2.3 Protocolos

A figura 6.2 demonstra os três módulos que compõem a arquitetura básica do sistema UbiquitOS. Conforme foi visto, cada módulo possui responsabilidades distintas e relaciona-se com os demais para execução de suas atividades. Como forma de padronizar o processo de comunicação entre módulos, foi definido um pequeno conjunto de protocolos que estabelecem regras para o diálogo e formatação das mensagens trocadas entre si.

Em síntese, um protocolo é definido para a comunicação entre os módulos `ubiquitos-provider` e `ubiquitos-smartspace` e dois outros são definidos para a comunicação entre `ubiquitos-smartspace` e `ubiquitos-client`, assim demonstrado na figura 6.21.

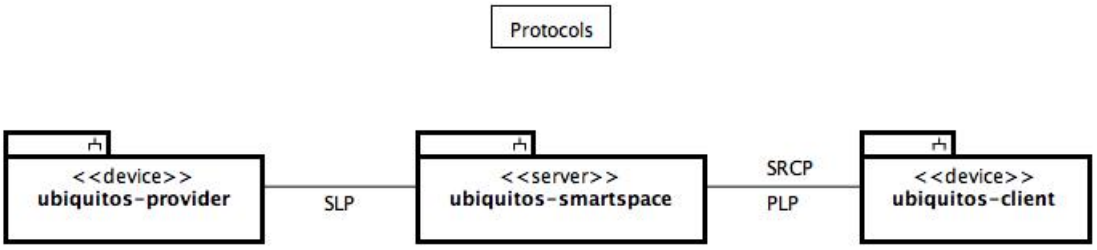

Figura 6.21: Protocolos do sistema UbiquitOS

6.2.3.1 SLP - *Service Listing Protocol*

O primeiro processo de comunicação realizado no âmbito do sistema UbiquitOS ocorre quando um dispositivo com suporte à tecnologia entra no *smart-space* e este lhe questiona sobre a lista de serviços por ele providos, para posterior publicação no ambiente (seções 6.2.1.1 e 6.2.1.2). À sequência de passos que então se desencadeia, deu-se o nome de *Service Listing Protocol*, ilustrado pela figura 6.22:

Para solicitar ao dispositivo (`ubiquitos-provider`) a listagem de seus serviços, o *smart-space* (`ubiquitos-smartspace`) envia, através da tecnologia de comunicação negociada, a mensagem REQUEST_SERVICE_LISTING. Ao receber esta mensagem, o dispositivo, que mantém na forma serializada, o *bytecode* dos *drivers* para comunicação com seus serviços, responde ao *smart-space* enviando-lhe inicialmente a quantidade de serviços disponíveis e, então, uma sequência que é repetida para todos os serviços providos pelo dispositivo, composta pelo nome do serviço, o tamanho, em bytes, de seu *driver* e a sequência de *bytes* que constituem o *driver*.

Para cada *driver* recebido, o *smart-space* o recria na forma de objeto de software e, através da biblioteca EasyJini, faz sua publicação na rede para que esteja disponível a outros dispositivos.

6.2.3.2 PLP - *Provider Listing Protocol*

Ao requerer a adaptação de um serviço, é apresentada ao usuário do dispositivo cliente uma interface de interação através da qual ele solicita a lista de

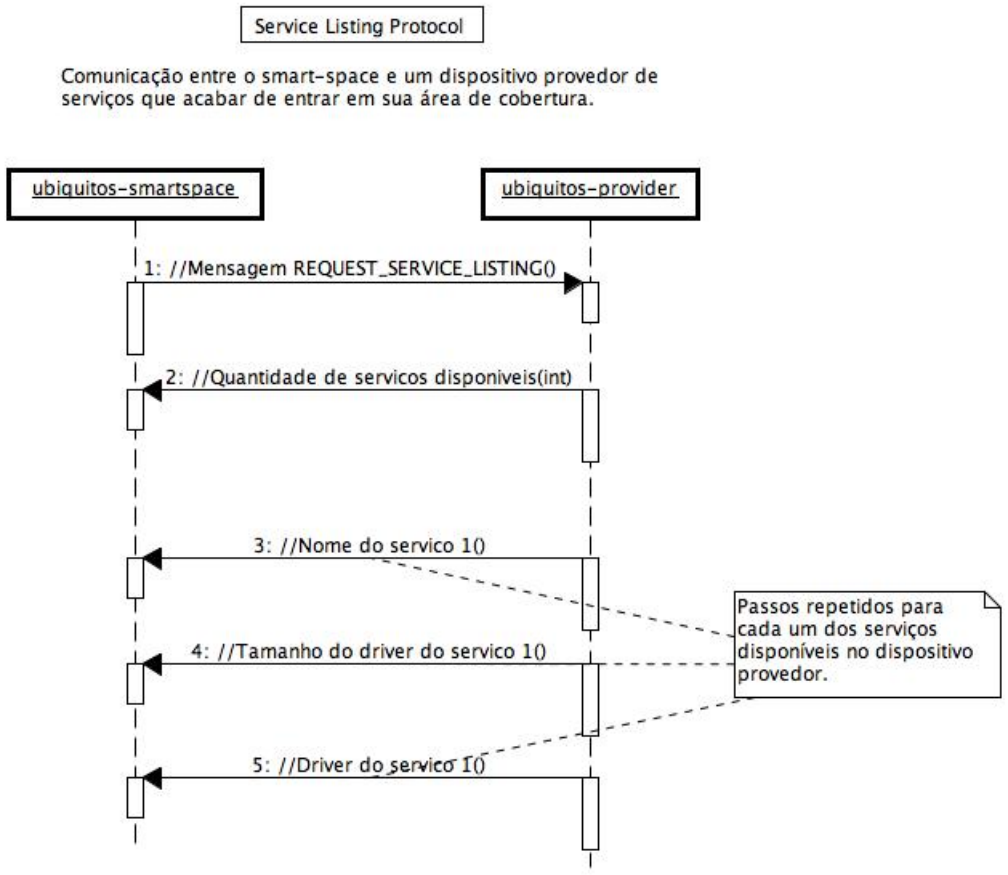

Figura 6.22: *Service Listing Protocol* para comunicação entre os módulos `ubiquitos-smartspace` e `ubiquitos-provider`.

provedores do serviço selecionado. Esta solicitação é processada pelo módulo `ubiquitos-client`, que a delega ao módulo `ubiquitos-smartspace`, conforme a figura 6.23.

Quando o usuário seleciona o serviço que deseja adaptar e solicita a listagem de provedores disponíveis no *smart-space*, a mensagem REQUEST_SERVICE_PROVIDERS é disparada pelo dispositivo cliente ao *middleware* do *smart-space* seguida pela descrição do tipo do serviço desejado. Ao recebê-la, o módulo `ubiquitos-smartspace` realiza uma varredura na rede do ambiente para a identificação de dispositivos provedores daquele tipo de serviço e transmite de volta, ao dispositivo, a quantidade de provedores localizados e suas respectivas identificações.

O dispositivo recebe esta lista e a apresenta ao usuário para sua seleção.

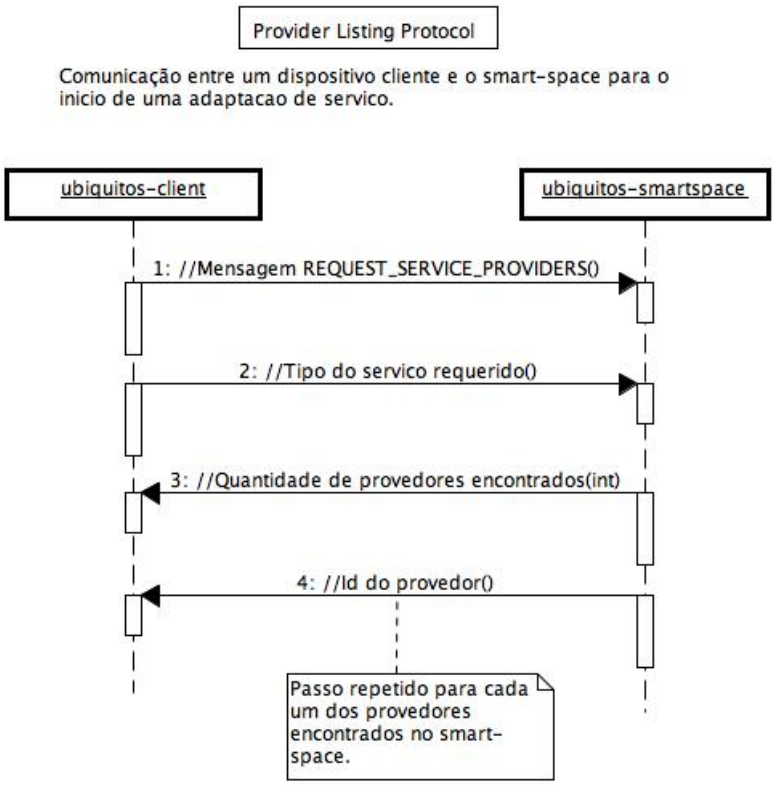

Figura 6.23: *Provider Listing Protocol* para o início da comunicação entre os módulos `ubiquitos-client` e `ubiquitos-smartspace`.

6.2.3.3 RSCP - *Remote Service Call Protocol*

Quando o usuário recebe a lista de provedores disponíveis para o serviço de sua escolha, com base em seus próprios interesses, ele seleciona o dispositivo que melhor lhe satisfaça. Neste ponto, é criado o conector cliente responsável por firmar a comunicação entre o aplicativo e o *driver* do serviço remoto localizado no provedor selecionado. Como dito na seção 6.2.2.1, o conector possui a mesma interface que o serviço requerido. Quando então da invocação local de seus serviços, a sequência de mensagens da figura 6.24 é iniciada.

Ao chamar localmente o serviço do conector cliente, o módulo `ubiquitos-client`, na figura do conector cliente, transmite ao módulo `ubiquitos-smartspace` uma mensagem indicando o tipo do serviço cuja conexão remota se deseja, acompanhado, na sequência, do identificador do provedor selecionado para o serviço. De posse dessas duas informações, o *smart-space* busca na rede o referido serviço e instancia um conector para intermediar o seu acesso. A partir daí, cada instrução enviada ao conector servidor é processada pelo *driver* do serviço e sua resposta devolvida ao conector cliente.

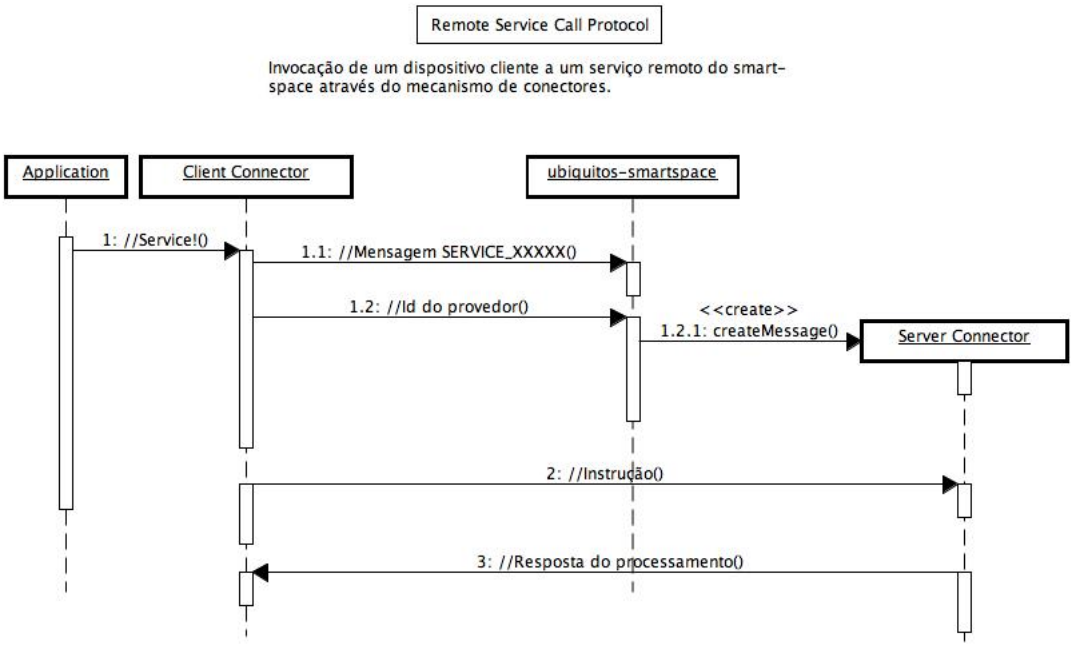

Figura 6.24: *Remote Service Call Protocol* para a invocação remota de serviços entre `ubiquitos-client` e `ubiquitos-smartspace`.

6.3 Modelo de Implementação

Esta seção apresenta detalhes do código desenvolvido para implementação do modelo de projeto da seção anterior. Nela, serão apresentados o ambiente de desenvolvimento utilizado e a listagem dos principais artefatos produzidos.

6.3.1 Ambiente de desenvolvimento

Seguem os detalhes do ambiente de desenvolvimento utilizado para construção da arquitetura proposta:

Hardware	Apple PowerPC G4 1.67 GHz 1 GB DDR2 SDRAM 512 KB L2 Cache
Sistema Operacional	MacOS X 10.4.9
Compilador	Java Development Kit 1.5.0-07
Ferramenta de *Build*	Apache Maven 2.0.5
IDE	NetBeans 5.0
Emulador de Dispositivos JME	mpowerplayer 2.0.898
Implementação da JSR 82	avetanaBluetooth

Tabela 6.2: Detalhes do ambiente de desenvolvimento

6.3.2 Artefatos de implementação

Além dos três módulos básicos da arquitetura UbiquitOS, duas bibliotecas auxiliares foram desenvolvidas, respectivamente para o suporte às tecnologias de comunicação Bluetooth e descoberta e registro de serviços em redes Jinis. As bibliotecas BtUtil [64] e EasyJini [65], além de fundamentos técnicos utilizados sob a arquitetura, tornaram-se projetos de código livre, disponíveis à toda comunidade e adotadas internacionalmente.

6.3.2.1 BtUtil

BtUtil [64] é a biblioteca desenvolvida para o suporte à comunicação Bluetooth. A listagem abaixo apresenta a estrutura de arquivos e diretórios de seu módulo base, chamado de btutil-base. Além deste, o projeto BtUtil é também composto por dois outros pacotes com exemplos de utilização do módulo básico em ambientes desktop (btutil-base-examples) e móvel (btutil-midp-examples). Os módulos btutil-base e btutil-base-examples foram estruturados segundo o padrão de pastas do Apache Maven, ferramenta utilizada para automatização do processo de build dos projetos. O módulo btutil-midp-examples, em contrapartida, foi totalmente desenvolvido através da ferramenta NetBeans e, por consequência, é sob sua estrutura padrão de pastas que se dispõem seus artefatos.

```
btutil-base/
    pom.xml
    src/main/java/br/unb/cic/bluetooth/BtUtil.java
    src/main/java/br/unb/cic/bluetooth/BtUtilClientAdapter.java
    src/main/java/br/unb/cic/bluetooth/BtUtilClientListener.java
    src/main/java/br/unb/cic/bluetooth/BtUtilException.java
    src/main/java/br/unb/cic/bluetooth/BtUtilServerListener.java

btutil-base-examples/
    pom.xml
    src/main/java/br/unb/cic/bluetooth/examples/ServiceRegistrationApp.java
    src/main/java/br/unb/cic/bluetooth/examples/ServiceDiscoveryApp.java
    src/main/java/br/unb/cic/bluetooth/examples/DeviceDiscoveryApp.java
    src/main/java/br/unb/cic/bluetooth/examples/GenericTest.java

btutil-midp-examples/
    build.xml
    nbproject/project.xml
    nbproject/project.properties
    nbproject/private
    nbproject/private/private.properties
    nbproject/private/private.xml
    nbproject/build-impl.xml
    nbproject/genfiles.properties
    src/br/unb/cic/bluetooth/midp/examples/ServiceDiscoveryMidlet.java
    src/br/unb/cic/bluetooth/midp/examples/DeviceDiscoveryMidlet.java
    src/br/unb/cic/bluetooth/midp/examples/ServiceRegistrationMidlet.java
    src/br/unb/cic/bluetooth/midp/examples/MyServiceDiscoveryMidlet.java
```

6.3.2.2 EasyJini

EasyJini [65] é a biblioteca desenvolvida para o suporte à publicação e descoberta de serviços em redes Jini [62]. Além de seu módulo básico, easyjini-base, foram também desenvolvidos códigos de exemplos de utilização da biblioteca, no módulo easyjini-base-examples.

```
easyjini-base/
    pom.xml
    src/main/java/br/unb/cic/jini/EasyJini.java
    src/main/java/br/unb/cic/jini/EasyJiniException.java

easyjini-base-examples/
    pom.xml
    src/main/java/br/cic/jini/example/SpeakersLocator.java
    src/main/java/br/cic/jini/example/MyDummyService.java
    src/main/java/br/cic/jini/example/Speaker.java
    src/main/java/br/cic/jini/example/EnglishSpeaker.java
    src/main/java/br/cic/jini/example/PortugueseSpeaker.java
    src/main/java/br/cic/jini/example/FrenchSpeaker.java
    src/main/java/br/cic/jini/example/SpeakersRegistration.java
    src/main/java/br/cic/jini/example/SimpleServiceLocator.java
    src/main/java/br/cic/jini/example/SimpleServiceRegistration.java
```

6.3.2.3 ubiquitos-smartspace

O projeto ubiquitos-smartspace corresponde à maior parcela do código desenvolvido. Nele, foram desenvolvidos todos os elementos de projeto descritos na seção 6.2.1.1. Sua estrutura de pastas segue o padrão adotado pelo Apache Maven.

```
ubiquitos/ubiquitos-smartspace/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos/Main.java
    src/main/java/br/unb/cic/ubiquitos/adaptabilityengine/AdaptabilityEngine.java
    src/main/java/br/unb/cic/ubiquitos/connectionmanager/BluetoothConnectionManager.java
    src/main/java/br/unb/cic/ubiquitos/devicemanager/DeviceManager.java
    src/main/java/br/unb/cic/ubiquitos/radar/Radar.java
    src/main/java/br/unb/cic/ubiquitos/radar/RadarListener.java
    src/main/java/br/unb/cic/ubiquitos/servicemanager/ServiceManager.java
    src/main/java/br/unb/cic/ubiquitos/servicemanager/connectors/SpeakerBluetoothServerConnector.java
    src/main/java/br/unb/cic/ubiquitos/servicemanager/entry/ServiceHost.java
    src/main/java/br/unb/cic/ubiquitos/util/JarUtil.java
    src/main/resources/log4j.properties
    src/main/resources/policy.all
```

6.3.2.4 ubiquitos-provider

O projeto ubiquitos-provider implementa o *design* da seção 6.2.1.2. Seu código foi distribuído em dois subprojetos: ubiquitos-provider-java e ubiquitos-provider-java-bluetooth. No primeiro (ubiquitos-provider-java), estão implementados todos os recursos básicos para criação de um provedor de serviços UbiquitOS baseados na tecnologia Java. O segundo (ubiquitos-provider-java-bluetooth), especialização do primeiro, disponibiliza também o suporte à tecnologia Bluetooth na criação de prvedores de serviços UbiquitOS.

```
ubiquitos/ubiquitos-provider-java/
    pom.xml
    src/site/site.xml
    src/main/java/br/unb/cic/ubiquitos/provider/java/UbiquitOSConnection.java
    src/main/java/br/unb/cic/ubiquitos/provider/java/factory/ProviderFactory.java
    src/main/java/br/unb/cic/ubiquitos/provider/java/SmartSpaceListener.java
    src/main/java/br/unb/cic/ubiquitos/provider/java/JavaServiceProvider.java

ubiquitos/ubiquitos-provider-java-bluetooth/
    pom.xml
    src/site/site.xml
    src/main/java/br/unb/cic/ubiquitos/provider/java/bluetooth/BluetoothConnection.java
    src/main/java/br/unb/cic/ubiquitos/provider/java/bluetooth/BluetoothServiceProvider.java
```

Os dois subprojetos acima constituem o núcleo básico da implementação de dispositivos provedores de serviços para ambientes UbiquitOS. Sobre este núcleo, constroem-se as especializações para cada dispositivo provedor de serviços. A listagem que segue representa a implementação do módulo ubiquitos-provider para o aparelho celular Nokia N80.

```
ubiquitos-ext/ubiquitos-provider-nokiaN80/
    pom.xml
    src/site/site.xml
    src/main/java/br/unb/cic/ubiquitos-ext/provider/nokia/n80/NokiaN80ServiceProvider.java
    src/main/java/br/unb/cic/ubiquitos-ext/provider/nokia/n80/NokiaN80UbiquitOSProvider.java
```

Para o estudo de caso descrito na seção 6.4, e com o objetivo de simular recursos distintos de diferentes dispositivos, foram criadas três implementações de um mesmo serviço Speaker. Essas implementações correspondem aos *drivers* de serviços exemplificados na seção 6.2.1.2. Para cada implementação de serviço (e.g. EnglishSpeaker), corresponde uma fábrica (e.g. EnglishSpeakerFactory) responsável por gerar sua representação serializada que será armazenada dentro da implementação do módulo ubiquitos-provider no dispositivo (e.g. NokiaN80ServiceProvider).

```
ubiquitos-ext/ubiquitos-provider-speaker-english/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/EnglishSpeaker.java
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/EnglishSpeakerFactory.java

ubiquitos-ext/ubiquitos-provider-speaker-french/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/FrenchSpeaker.java
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/FrenchSpeakerFactory.java

ubiquitos-ext/ubiquitos-provider-speaker-portuguese/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/PortugueseSpeaker.java
    src/main/java/br/unb/cic/ubiquitos-ext/provider/speaker/PortugueseSpeakerFactory.java
```

6.3.2.5 ubiquitos-client

O projeto ubiquitos-client corresponde à implementação da seção 6.2.1.3. No subprojeto ubiquitos-client-commons, está disponibilizado o fluxo básico de telas e operações para busca e seleção de novos provedores de um serviço escolhido. Todo o projeto foi desenvolvido sobre a IDE NetBeans 5.0.

```
ubiquitos/ubiquitos-client/ubiquitos-client-commons/
    build.xml
    nbproject/project.xml
    nbproject/project.properties
    nbproject/private/private.properties
    nbproject/private/private.xml
    nbproject/build-impl.xml
    nbproject/genfiles.properties
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClientControllerListener.java
    src/br/unb/cic/ubiquitos/client/commons/connectors/SpeakerBluetoothClientConnector.java
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClient.java
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClientController.java
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClientListener.java
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClientGUI.java
    src/br/unb/cic/ubiquitos/client/commons/UbiquitOSClientGUI.mvd
```

Já o subprojeto ubiquitos-client-nokiaN80 implementa o módulo ubiquitos-client da arquitetura UbiquitOS para o aparelho celular Nokia N80, tendo por base os recursos da implementação ubiquitos-client-commons.

```
ubiquitos-ext/ubiquitos-client/ubiquitos-client-nokiaN80/
    build.xml
    nbproject/project.xml
    nbproject/project.properties
    nbproject/private/private.properties
    nbproject/private/private.xml
    nbproject/build-impl.xml
    nbproject/genfiles.properties
    src/br/unb/cic/ubiquitos/client/nokiaN80/UbiquitOSClient.java
    src/br/unb/cic/ubiquitos/client/nokiaN80/Main.java
    src/br/unb/cic/ubiquitos/client/nokiaN80/Main.mvd
```

6.3.2.6 ubiquitos-commons

No projeto ubiquitos-commons foram inseridas todas as classes de uso comum entre os módulos ubiquitos-smartspace, ubiquitos-provider e ubiquitos-client da arquitetura UbiquitOS.

O projeto foi fragmento em três partes. Em ubiquitos-commons estão exceções e biblioteca de constantes genéricas para os três módulos. Em ubiquitos-services estão as descrições dos serviços suportados pela arquitetura. Ou seja, para cada serviço implementado pelo módulo ubiquitos-provider, deve corresponder uma respectiva interface Java no módulo ubiquitos-commons/ubiquitos-services. O ubiquitos-services-serializable contém as mesmas interfaces do módulo ubiquitos-services acrescidas da capacidade de serem serializadas. Dessa forma, o módulo ubiquitos-services é usado pelos dispositivos que não suportam serialização de objetos, como aqueles desenvolvidos em MIDP, e o módulo ubiquitos-service-serializable é usado pelos dispositivos com suporte e exigências quanto à serialização de objetos.

```
ubiquitos/ubiquitos-commons/
    pom.xml

    ubiquitos-commons/
```

```
    pom.xml
    src/main/java/br/unb/cic/ubiquitos/commons/UbiquitOSException.java
    src/main/java/br/unb/cic/ubiquitos/commons/UbiquitOSUUIDs.java
    src/main/java/br/unb/cic/ubiquitos/commons/UbiquitOSMessages.java

ubiquitos-services/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos/commons/services/listeners/MouseListener.java
    src/main/java/br/unb/cic/ubiquitos/commons/services/Speaker.java
    src/main/java/br/unb/cic/ubiquitos/commons/services/Mouse.java

ubiquitos-services-serializable/
    pom.xml
    src/main/java/br/unb/cic/ubiquitos/commons/services/serializable/Speaker.java
    src/main/java/br/unb/cic/ubiquitos/commons/provider/factory/ProviderFactory.java
```

6.4 Estudo de Caso

Como já mencionado durante a descrição do Connection Manager, na seção 6.2.1.1, toda a arquitetura foi exercitada através de um simples serviço, de caráter apenas demonstrativo da proposta. Este serviço, especificado pela interface *Speaker*, define uma operação *compliment* através da qual uma saudação é emitida. Dessa forma, a especificação do serviço foi implementada por 3 dispositivos distintos, conforme mostra a figura 6.25. Foram utilizados para os testes, os aparelhos celulares Nokia modelos N80 e 6600 e um *laptop* de marca Machintosh, modelo PowerBook G4. A implementação do serviço *Speaker* por cada dispositivo produz resultados distintos, na forma de saudações, também vistos na figura.

A figura 6.26 apresenta a distribuição dos diversos componentes da arquitetura entre os dispositivos participantes do estudo de caso.

Em cada um dos três dispositivos (Nokia N80, Nokia 6600 e PowerBook) foi implantado o módulo `ubiquitos-provider`, responsável pela disponibilização da implementação local do serviço *Speaker* (NokiaN80Speaker, Nokia6600Speaker e PowerBookSpeaker) ao ecosistema do *smart-space*. Como já visto, esta disponibilização se dá na forma de *drivers*, de cada serviço, que são mantidos serializados pelos módulos `ubiquitos-provider`.

Dessa forma, todos os dispositivos foram submetidos à área de cobertura do *smart-space*, onde, graças ao módulo `ubiquitos-smartspace` da arquitetura, tiveram seus respectivos *drivers*, para acesso às implementações do serviço *Speaker*, publicados.

Para realização dos testes, uma pequena aplicação de exemplo foi desenvolvida no dispositivo Nokia N80, com a simples funcionalidade de invocação do serviço *Speaker*. Assim, originalmente, ao se executar o aplicativo de teste, a mensagem "Olá do Nokia N80" era visualizada, resultante da invocação da implementação local do serviço *Speaker* (NokiaN80Speaker).

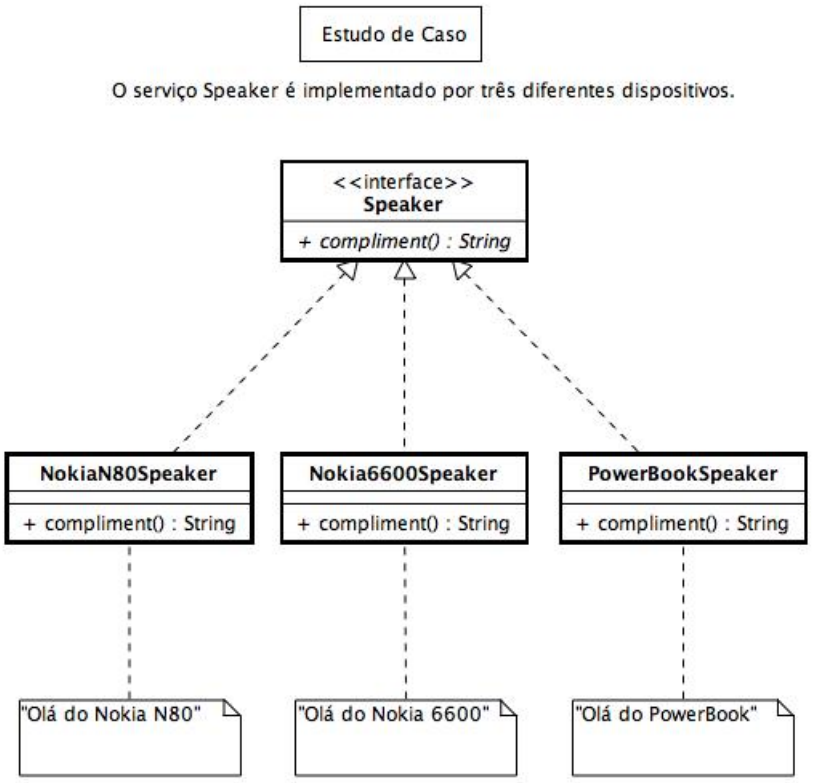

Figura 6.25: Diferentes implementações para o serviço Speaker.

Para permitir a adaptação de serviços neste dispositivo, foi também implantado no Nokia N80 o módulo `ubiquitos-client`, responsável pela possibilidade de acesso de um dispositivo aos serviços dos demais. Com isso, agregou-se à aplicação de teste a possibilidade de alterar a implementação do serviço *Speaker* então utilizada.

Assim, ao iniciar o teste, o usuário executa a aplicação, obtendo a mensagem "Olá Nokia N80". Em seguida, conforme descrito na seção 6.2.1.3, o usuário solicita a troca do provedor do serviço *Speaker*. O processo, inicialmente, busca na rede pela lista de provedores disponíveis para o referido serviço, obtendo a identificação dos dispositivos Nokia 6600 e PowerBook. O usuário seleciona um destes dispositivos (provedores) e, deste ponto em diante, a execução do aplicativo de teste resultará nas mensagens "Olá do Nokia 6600" ou "Olá do PowerBook", conforme dispositivo escolhido.

6.5 Resultados

Indiretamente, o projeto UbiquitOS disponibilizou à comunidade duas bilbiotecas [64, 65] de código livre para o desenvolvimento de aplicações com suporte, respectivamente, às tecnologias Bluetooth e Jini.

Em síntese, a construção de uma arquitetura de *middleware* para a adaptabilidade de serviços em sistemas de computação ubíqua representa contribuições diretas aos usuários desses sistemas, aos desenvolvedores de *middleware* para *smart-spaces* e às pesquisas em gerência de recursos computacionais.

Aos futuros usuários dos sistemas de computação ubíqua com aderência a esta proposta, fica reservado o direito, a qualquer tempo, de substituição do dispositivo em uso no *smart-space* por outro de igual funcionalidade, seja por razões de qualidade do serviço prestado, localização privilegiada do dispositivo eleito ou mesmo problemas técnicos do dispositivo original. Isso mantém a satisfação dos integrantes do sistema, cujas vidas são facilidadas pela informática através utilização mais intensa das potencialidades do ambiente.

Para os responsáveis pelo desenvolvimento de soluções em software de *smart-spaces*, é facilitada a incorporação da adaptação de serviços em suas arquiteturas sem o esforço de *design* integral de um novo mecanismo, reduzindo os custos do projeto e enriquecendo assim o produto com funcionalidades de valor para seus usuários.

Por outro lado, além de servir para compor arquiteturas pré-existentes, este projeto também pode ser visto como fundação básica para o desenvolvimento de outras pesquisas relacionadas ao campo da computação ubíqua. O sistema desenvolvido, se acrescido de recursos de análise das informações de contexto, identificação e localização de entidades do sistema e pró-atividade para a reconfiguração automática, pode constituir-se em uma completa solução de *middleware* para *smart-spaces*.

Em termos de gerência de recursos, este trabalho sugere a adaptabilidade de serviços como ferramenta singular contra o desperdício de recursos computacionais em ambientes de computação ubíqua, conforme ilustrado no capítulo 5. A possibilidade de substituição de um provedor de serviço por outro provedor, aliada a uma estrutura modular e programável da arquitetura, garante que sempre haverá a possibilidade de utilização de um dispositivo presente no ambiente, evitando assim sua ociosidade e o eminente desperdício de seus recursos computacionais.

Em relação às demais iniciativas da comunidade, a proposta UbiquitOS apresenta relevantes contribuições pelo estudo sistemático da adaptabilidade de servi-

ços na computação ubíqua. Para melhor enquadramento deste trabalho no escopo das pesquisas mais recentes, um quadro comparativo entre as características da adaptabilidade de serviços de cada projeto e da presente proposta é apresentado na tabela 6.3. As informações contidas na tabela representam conclusões obtidas a partir das publicações citadas ao longo dos capítulos anteriores.

	Aura	Gaia	Gator Tech	ISAM	EasyLiving	**UbiquitOS**
Abordagem sistemática da adaptabilidade de serviços	-	-	-	-	-	x
Facilidade de suporte a novos dispositivos	-	-	x	x	x	x
Integração espontânea de dispositivos	-	-	x	-	x	x

Tabela 6.3: Quadro comparativo entre projetos da *ubicomp*

Nas seções que seguem, são descritos os apontamentos de cada tópico do quadro comparativo da tabela 6.3.

6.5.1 Abordagem sistemática da adaptabilidade de serviços

Entende-se por abordagem sistemática da adaptabilidade de serviços o isolamento dos mecanismos atuantes na viabilização da adaptação de serviços e seu estudo individualizado.

Pelo observado nos projetos Aura, Gaia, ISAM, EasyLiving e Gator Tech, todos, de forma direta ou indireta referem-se à adaptabilidade de serviços como um benefício de suas propostas mas em nenhum deles a propriedade é explorada de forma tão explícita quanto a descrição apresentada no capítulo 5.

6.5.2 Facilidade de suporte a novos dispositivos

A facilidade de suporte a novos dispositivos busca identificar o quão flexível é a arquitetura para a expansão do ecosistema do *smart-space*.

Nos artigos do projeto Aura, não há menções sobre a agregação de novos dispositivos mas apenas a possibilidade de encapsulamento de aplicações tradicionais (MSWord, emacs, Notepad) para sua incorporação ao sistema [23].

No projeto Gaia, há a necessidade de suporte de um ORB em cada dispositivo membro do espaço, o que reduz sobremaneira as possibilidades de novas afiliações.

[4] diz que uma das motivações originárias do projeto Gator Tech House foi o desenvolvimento de um ambiente pervasivo dinâmico, capaz de evoluir e se adaptar às tecnologias emergentes, ao contrário das primeiras gerações de sistemas

ubíquos, estáticos no tempo e extremamente frágeis à adoção de novos padrões e novos dispositivos.

O projeto ISAM também não aborda este assunto diretamente mas, por se tratar de uma arquitetura genérica para ambientes de computação móvel, acredita-se que este atributo é internamente contemplado.

Apesar de não descrever os detalhes da implementação, a documentação do projeto EasyLiving o defende como uma arquitetura para possibilitar a agregação dinâmica de diversos dispositivos de I/O para enriquecimento da experiência de uso do ambiente pelo usuário.

Por fim, o projeto UbiquitOS, graças a uma arquitetura de fácil extensão, através dos módulos `ubiquitos-provider` e `ubiquitos-client`, garante que qualquer novo dispositivo seja capaz de se integrar à sua comunidade de provedores e clientes de serviços.

6.5.3 Integração espontânea de dispositivos

Pela integração espontânea de dispositivos deseja-se compreender o quão burocráticas são as arquiteturas para a comunicação entre dois elementos do ecosistema de dispositivos de seus *smart-spaces*. Foram marcados na tabela apenas os projetos onde o processo fosse o mais automatizado possível.

Os projetos Aura e ISAM não trazem qualquer destaque para este item.

Já o projeto Gaia, em contrapartida, com a proposta do barramento unificado para a comunicação entre objetos, defende a possibilidade de integração de dispositivos de diferentes fabricantes, com diferentes sistemas operacionais e diferentes implementações de *middleware*. Não há, entretanto, qualquer alusão quanto à espontaneidade do processo.

Novamente sem trazer detalhes específicos, o projeto Easy Living diz definir um conjunto de tecnologias destinadas à integração dinâmica de dispositivos.

Através da *Sensor Platform Layer*, o projeto Gator Tech também viabiliza a integração espontânea de dispositivos, graças a publicação dinâmica dos *drivers* necessários à sua utilização.

E, por basear-se na proposta da carga em tempo de execução de seus *drivers*, o projeto UbiquitOS acompanha o projeto Gator Tech House no gozo da integração espontânea de dispositivos.

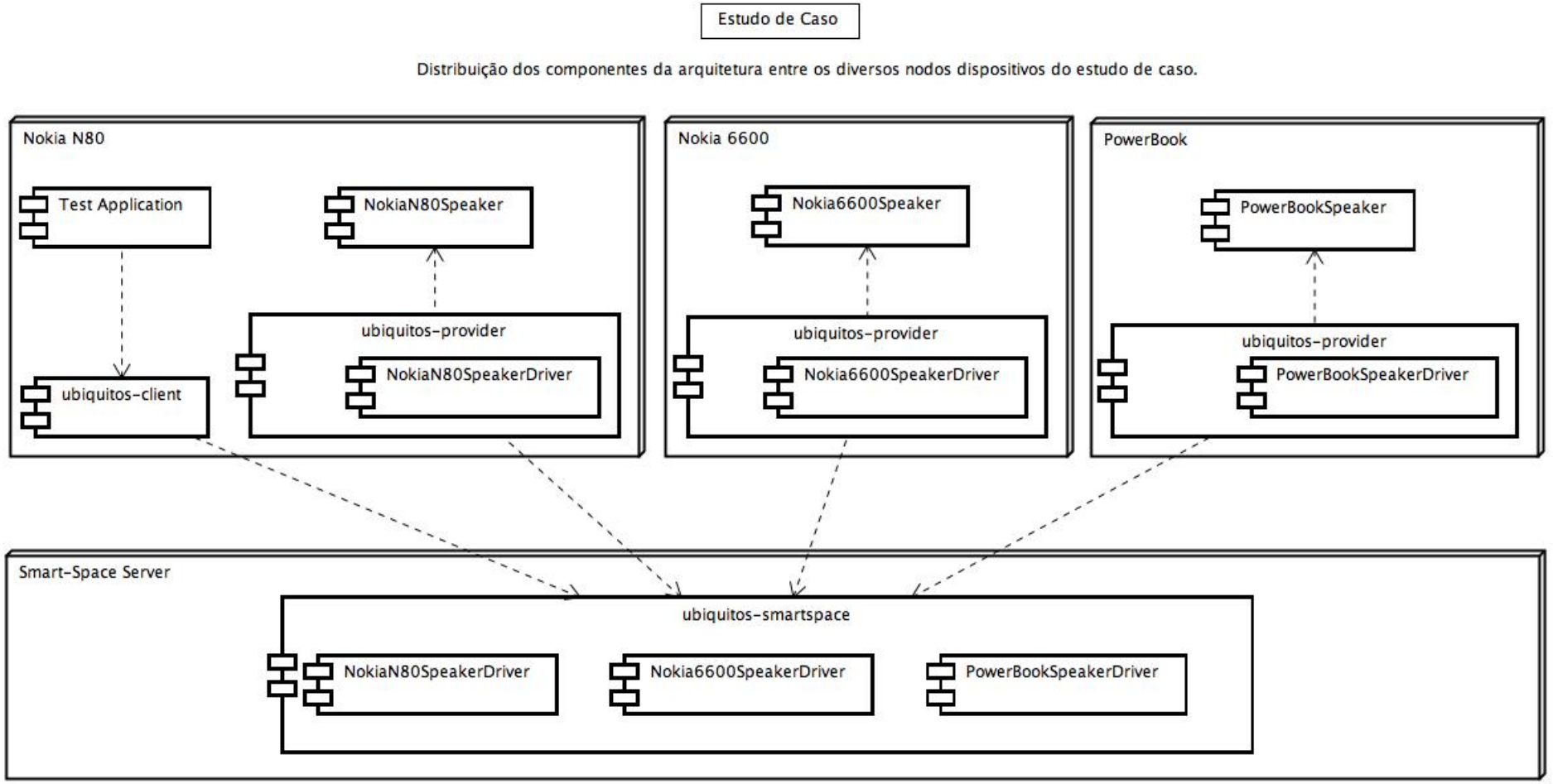

Figura 6.26: Distribuição de componentes entre os dispositivos da arquitetura.

Capítulo 7

Conclusões

"A tese é como um porco: nada se desperdiça."
Umberto Eco, 1995

Por inexistir o senso comum sobre a consolidação de um grande problema a ser mitigado na *ubicomp*, o presente trabalho vem à tona tendo a adaptabilidade de serviços como pilar de suas investigações. Apesar de já contemplada, ainda que indiretamente, por vários outros projetos (capítulo 4), sua evolução potencializa grandes benefícios à computação ubíqua (capítulo 5) e, por esta razão, o tema tornou-se a base de sustentação para o desenvolvimento do projeto UbiquitOS.

Esta pesquisa apresentou uma proposta de arquitetura de *middleware* para a adaptabilidade de serviços em sistemas de computação ubíqua. O estudo foi iniciado, no capítulo 2, através da exploração da essência deste novo paradigma computacional, onde pôde-se observar as expectativas da comunidade científica em torno do assunto. Para compreender o sentido dos trabalhos mais recentes, foram enumerados, no caítulo 3, alguns projetos de relevância da *ubicomp*, a partir dos quais verificou-se a ausência de um direcionamento comum dos atuais esforços. Por conseguinte, a adaptabilidade de serviços foi a propriedade dos sistemas da computação ubíqua selecionada para estudo, cuja definição foi apresentada no capítulo 4. O capítulo 5 é iniciado com a análise de um potencial problema de desperdício de recursos computacionais em *smart-spaces* e sugere a adaptabilidade de serviços como possível estratégia para sua resolução, através de uma arquitetura modular que permita o compartilhamento de recursos entre dispositivos em *middlewares* de sistemas ubíquos. O detalhamento do projeto desta arquitetura, batizada de UbiquitOS, foi, por fim, apresentado no capítulo 6.

Os resultados e contribuições desta proposta estão descritos na seção 6.5. Como destaques da solução, pode-se apontar a abordagem sistêmica da adaptabilidade de serviços, a possibilidade de expansão do ecosistema e a integração

espontânea entre dispositivos.

A abordagem sistêmica da adaptabilidade de serviços em uma arquitetura para *smart-spaces*, demonstrada através do isolamento de seus mecanismos provedores, possibilita o fino ajuste de seu comportamento, favorecendo a identificação de fragilidades e o desenvolvimento de técnicas específicas para maior robustez, escalabilidade, segurança e flexibilidade da adaptação de serviços.

Graças à sua infra-estrutura programável, pela API dos módulos `ubiquitos-provider` e `ubiquitos-client`, que fornece toda a base para publicação e descoberta de serviços e dispositivos, é possível garantir a expansão constante do ecosistema de dispositivos do sistema, através do desenvolvimento de novos *drivers* de serviços sobre a API disponibilizada.

A integração espontânea de dispositivos é garantida pelo mecanismo de conectores, que permite que, uma vez desenvolvidos os módulos provedor e cliente de dois dispositivos, sua comunicação flua de forma dinâmica, natural e expontânea, sem a necessidade de qualquer intervenção do usuário, seja para o estabelecimento de regras ou para a instalação de softwares adicionais.

Entretanto, alguns pontos da arquitetura ainda merecem cuidados especiais, a fim de que a proposta atinja elevados níveis de maturidade e adoção pela comunidade de pesquisa, como a dependência da plataforma Java, a conectividade limitada à tecnologia Bluetooth e a não tolerância a falhas pelos protocolos implementados.

Como pré-definido na seção 5.2.7, a tecnologia Java foi utilizada como base de todo o desenvolvimento. Todavia, apesar de interessante em termos de sua alta robustez, segurança e suporte a múltiplas plataformas, ainda não é real sua ubiquidade no parque de dispositivos disponíveis no mercado, o que marginaliza do ecosistema um ou outro dispositivo que não suporte a tecnologia.

Também na seção 5.2.7 ficou definida a tecnologia Bluetooth como base da comunicação do sistema que, embora tenha tendências favoráveis à sua democratização, possui restrições de mercado bem conhecidas, que lhe reservam apenas uma pequena fatia de suporte no universo de dispositivos atualmente disponíveis. Ou seja, excluem-se deste *smart-space* celulares, *laptops* e computadores de mão sem suporte a redes Bluetooth, bem como demais dispositivos embarcados com etiquetas RFID ou outra tecnologia de comunicação remota qualquer.

Por terem sido concebidos em ambiente controlado, os protocolos descritos na seção 6.2.3 são frágeis em casos de ocorrência de falhas de comunicação no sistema. Apesar de a infra-estrutura Jini, base para os mecanismos de publicação e descoberta de serviços, ser robusta frente à inconstância da rede, possíveis

problemas podem ocorrer quando a execução ultrapassa seu ambiente de execução, como, por exemplo, nos instantes de troca de dados do *smart-space* com os dispositivos do ambiente.

Por fim, este trabalho plantou a semente da computação ubíqua no Departamento de Ciência da Computação da Universidade de Brasília. A partir dele, inúmeros projetos e novas pesquisas podem evoluir. Segue abaixo uma breve listagem de lacunas candidatas aos trabalhos futuros desta seara:

- Complementar a atual arquitetura para construção de um *middleware* completo para *smart-spaces*, através da implementação de gerência de contexto, autenticação e autorização de usuários, controle de concorrência, e demais atributos comuns em *middlewares* para *smart-spaces*;
- Adotar tecnologia independente de linguagem de programação para descrição e acesso a serviços, a fim de que dispositivos de diferentes gerações e fabricantes possam integrar-se à comunidade do sistema;
- Dar suporte a outras tecnologias de conectividade, pelos mesmos motivos do item anterior;
- Incorporar tolerância a falhas aos protocolos para garantir maior transparência e invisibilidade das operações para os usuários finais do sistema;
- Facilitar a incorporação de novos serviços na arquitetura, elevando ao máximo o potencial de expansão do ecosistema;
- Implementar mecanismo de carga dinâmica de *drivers*, eliminando a necessidade dos conectores, para simplificar a arquitetura e agregar-lhe melhor desempenho;
- Implementar a adaptação automática de serviços, para uma maior sinergia da proposta com os princípios de pró-atividade da *ubicomp*;
- Utilizar ontologias para classificação dos serviços do ambiente, com o intuito de consolidação de um vocabulário comum e de fácil acesso a futuros desenvolvedores;
- Utilizar camada de conhecimento de Allemand [68] para *matching* de serviços, que pode auxiliar no processo de implementação da adaptação automática;

- Viabilizar a conectividade entre dispositivos, mesmo quando não houver conexão direta entre eles, também em prol da invisibilidade e pró-atividade do sistema;
- Desenvolver outros casos de uso (e.g. um celular utilizar a câmera de outro, controlar o ponteiro do mouse através do celular, etc), para a descoberta de cenários ainda não previstos e, consequentemente, maior amadurecimento da proposta;
- Expandir a pesquisa, através da análise detalhada de outros projetos, descritos na seção 3.6.

Referências Bibliográficas

[1] Kalle Lyytinen and Youngjin Yoo. Issues and challenges in ubiquitous computing. *Communications of the ACM*, 45(12):63–65, December 2002.

[2] Jo ao Sousa and David Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43, Montreal, Canada, Aug 2002.

[3] University of Illinois at Urbana-Champaign. Gaia - Active Spaces for Ubiquitous Computing. `http://gaia.cs.uiuc.edu` - Acessado em 25/02/2007.

[4] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The gator tech smart house: A programmable pervasive space. *IEEE Computer*, pages 50–60, March 2005.

[5] Adenauer Corrêa Yamin, Jorge Luis Victória Barbosa, Iara Augustin, and Cláudio Fernando Resin Geyer. ISAM: A software architecture for pervasive computing. *CLEI Electronic Journal*, 8(1), 2005.

[6] Mark Weiser and John Seel. Brown. A revised version of: The coming age of calm technology. *PowerGrid Journal 1.0*, 1.01, 1996. The important waves of technological change are those that fundamentally alter the place of technology in our lives.

[7] Mark Weiser. Ubiquitous computing. `http://www.ubiq.com/hypertext/weiser/UbiHome.html` – Acessado em 25/02/2007.

[8] Mark Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, October 1993.

[9] Mark Weiser. The world is not a desktop. *Perspectives article for ACM Interactions*, November 1993.

[10] D. Norman. *The Invisible Computer*. MIT Press, 1998.

[11] Ouahiba Fouial, Katia Abi Fadel, and Isabelle Demeure. Adaptive service provision in mobile computing environment. In *in Proc. 4th Proceedings 4th IEEE Int'l Conf. Mobile Wireless Communication Networks*, Set 2002.

[12] Xerox. Palo alto research center. Web Site, 2005.

[13] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

[14] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001.

[15] David Garlan et al. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 2002.

[16] Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE PERVASIVE Computing*, pages 70–81, JANUARY-MARCH 2002.

[17] Michael Beigl, Hans-W. Gellersen, and Albrecht Schmidt. Mediacups: Experience with design and use of computer-augmented everyday objects. *Computer Networks*, 35:401–409, March 2001.

[18] Mark Weiser and John Seely Brown. Designing calm technology. *PowerGrid Journal*, 1996. `http://nano.xerox.com/hypertext/weiser/calmtech/calmtech.htm` – Acessado em 26/02/2007.

[19] Paul Dourish. What we talk about when we talk about context. *Personal Ubiquitous Computing*, 8:19–30, 2004.

[20] Gregori D. Abowod and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, March 2000.

[21] Jo ao Pedro Sousa and David Garlan. From computers everywhere to tasks anywhere: The aura approach. Submitted for publication, April 2001.

[22] Carnegie Mellon University. Project aura - distraction-free ubiquitous computing. `http://www.cs.cmu.edu/~aura` – Acessado em 25/02/2007.

[23] Jo ao Pedro Sousa and David Garlan. The aura software architecture: an infrastructure for ubiquitous computing. Technical Report CMU-CS-03-183,

School of Computer Science, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890, August 2003.

[24] Renato Cerqueira, Christopher K. Hess, Manuel Román, and Roy H. Campbell. Gaia: A development infrastructure for active spaces. *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, 2001.

[25] OMG. CORBA. `http://www.corba.org` - Acessado em 22/02/2007.

[26] Microsoft. COM: Component Object Model Technologies. `http://www.microsoft.com/com` - Acessado em 22/02/2007.

[27] Sun Microsystems. Enterprise JavaBeans Technology. `http://java.sun.com/products/ejb` - Acessado em 22/02/2007.

[28] F. Bushmann et al. *Pattern-Oriented Software Architecture - A System of Patterns*, volume 1. Wiley, wiley edition, 2001.

[29] Free Software Foundation. BASH. `http://www.gnu.org/software/bash` - Acessado em 25/02/2007.

[30] OSGi Alliance. OSGi – the dynamic module system for java. `http://www.osgi.org` - Acessado em 25/02/2007.

[31] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. Easyliving: Technologies for intelligent environments. Technical report, Microsoft, 2000.

[32] UC Berkeley Computer Science Division. The ninja project - enabling internet-scale services from arbitrarily small devices. `http://ninja.cs.berkeley.edu` – Acessado em 25/02/2007.

[33] Electrical Engineering and Berkeley Computer Science Department at University of California. The endeavour expedition: Charting the fluid information utility. `http://endeavour.cs.berkeley.edu` - Acessado em 25/02/2007.

[34] Byung Y. Sung, Mohan Kumar, Behrooz Shirazi, and Swaroop Kalasapur. A formal framework for community computing. Technical report, University of Texas at Arlington, March 2003.

[35] MIT Laboratory for Computer Science and MIT Artificial Intelligence Laboratory. Mit project oxygen: Pervasive , human-centered computing. http://www.oxygen.lcs.mit.edu/index.html - Acessado em 25/02/2007.

[36] MIT Media Lab. Things that think. http://ttt.media.mit.edu - Acessado em 25/02/2007.

[37] Department of Computer Science & Engineering of University of Washington. Portolano: An expedition into invisible computing. http://portolano.cs.washington.edu - Acessado em 25/02/2007.

[38] Daniel Cheah, Ben Hendrickson, Janet Davis, Robert Grimm, Eric Lemar, Adam MacBeth, Steve Swanson, Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, and David Wetherall. one.world. http://one.cs.washington.edu - Acessado em 25/02/2007.

[39] Internet and Mobile Systems Laboratory at Hewlett Packard Laboratories. Cooltown. http://cooltown.hp.com - Acessado em 25/02/2007.

[40] iroom project. http://swig.stanford.edu/public/projects/iroom.

[41] Georgia Institute of Technology. Future computing environments. http://www.cc.gatech.edu/fce - Acessado em 25/02/2007.

[42] Next wave technologies and markets - interface. http://www.nextwave-interface.org.uk - Acessado em 25/02/2007.

[43] Engineering and Physical Sciences Research Council. Equator. http://www.equator.ac.uk - Acessado em 25/02/2007.

[44] University of Illinois at Urbana-Champaign. A component-based network-centric operating system for the next millennium. http://choices.cs.uiuc.edu/2k - Acessado em 25/02/2007.

[45] Integrated project on pervasive gaming. http://iperg.sics.se - Acessado em 25/02/2007.

[46] Engineering and Physical Sciences Research Council. A uk ubiquitous computing network. http://www-dse.doc.ic.ac.uk/Projects/UbiNet – Acessado em 25/02/2007.

[47] Ubiquitous computing evaluation consortium. http://www.ubiqcomputing.org – Acessado em 25/02/2007.

[48] Markus C. Huebscher and Julie A. McCann. Using real-time dependability in adaptive service selection. *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-icns'05)*, page 76, 2005.

[49] Markus Huebscher and Julie McCann. An adaptive middleware framework for context aware applications. *Personal Ubiquitous Computing*, 2006.

[50] Jo ao Pedro Sousa, Vahe Poladian, David Garlan, and Bradley Schmerl. Capitalizing on awareness of user tasks for guiding self-adaptation. In *International Workshop on Adaptive and Self-Managing Enterprise Applications*, pages 83–96, Porto, Portugal, 2005. 17th Conference on Advanced Information Systems Engineering.

[51] Seng W. Loke and Sea Ling. Analyzing observable behaviours of device ecology workflows analyzing observable behaviours of device ecology workflows. *Proceedings of the 6th International Conference on Enterprise Information Systems*, 2004.

[52] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[53] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *International Journal of Supercomputer Applications*, 15(3), 2001.

[54] Infrared data association. `http://www.irda.org` – Acessado em 25/02/2007.

[55] Bluetooth.org - the official bluetooth membership site. `http://www.bluetooth.org` – Acessado em 25/02/2007.

[56] Andrew Fano and Anatole Gershman. The future of business services in the age of ubiquitous computing. *Communications of the ACM*, 45(12):83–87, December 2002.

[57] Leonard M. Jessup and Daniel Robey. The relevance of social issues in ubiquitous computing environments. *Communications of the ACM*, 45(12):88–91, December 2002.

[58] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 2004.

[59] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, 2006.

[60] Remote method invocation home. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp – Acessado em 25/02/2007.

[61] Jxta. http://www.jxta.org – Acessado em 25/02/2007.

[62] The community resource for jini technology. http://www.jini.org – Acessado em 25/02/2007.

[63] Universal description, discovery and integration. http://uddi.org – Acessado em 25/02/2007.

[64] Alexandre Gomes. Btutil – bluetooth made easy. https://btutil.dev.java.net – Acessado em 25/02/2007.

[65] Alexandre Gomes. Easyjini – jini made easy. https://easyjini.dev.java.net – Acessado em 25/02/2007.

[66] Bluetooth specification v1.2. https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2 – Acessado em 25/02/2007.

[67] June Jang, KwangHee Lee, and Hoon Choi. The design and implementation of the surrogate system for j2me-devices. *IEEE*, pages 168–171, 2004.

[68] José Nelson Costa Allemand. Serviço baseado em semântica para descoberta de recursos em grade computacional. Master's thesis, Departamento de Ciência da Computação, Universidade de Brasília, Brasília, Distrito Federal, Brasil, 2006.